14 August 1991

C&R Systems
P.O. Box J
Blue Ridge Summit, PA 17214
BBS 717-765-8623

1. Congratulations on your purchase of BBS Express! ST. Since its release in 1987, BBS Express! ST has maintained a good reputation as a stable and easy to operate Bulletin Board System for the Atari ST line of computers.

2. The older ICD Inc. version 1.30 of the manual has been provided while the newer version is being prepared. You are on the list to automatically receive the new version of the manual when it is completed.

3. Please read the "README.1ST" file on Disk 1 and the manual thoroughly. They will provide you information on the improvements since version 1.30.

4. You have been provided with VERSION 1.72a of BBS Express! ST, SERIAL NUMBER Ø81416.

BBS Express! ST - Table of Contents

o Introduction 1

o The importance of registering your product with us . 1

o Terms used throughout this manual 1

o Getting Started 3

o Backing up your master diskette 3

o How Express! allocates its data files . . . 3

o Determining the best setup for your system . . 3

o Creating your Userlog database and message bases . . 4

o Loading the CREATOR program 4

o Setting up your userlog 4

o Setting up your message bases 4

o Creating the SYSDATA file for your system 6

o What is the SYSDATA file? 6

o Loading the SYSEDIT program 6

o The miscellaneous variables 7

o The message base definitions 11

o The file SIG area definitions 12

o The main command definitions 13

o The message base command definitions 14

o The Help file menus 15

o What are help files? 15

o The MENU option for getting user input . . . 17

o The SURVEY option for taking user surveys . . . 19

- **The Script file language 22**
  - An introduction 22
  - Command overview 25
  - Alphabetical script language command list (with page numbers) . . . 26
  - Defining your variables 28
  - Console Input / Output 29
  - File Input / Output 32
  - IF statement processing 38
  - Flow control 42
  - Miscellaneous commands 46
  - Referencing your variables as arrays 52
  - Using the dynamic table 55
  - VT52 color commands 57
  - Word parser commands 58
  - Some sample script files 59
  - The ENDCALL script file 64
  - The off-line script utilities 66
- **Running the BBS 67**
  - Loading the BBS program 67
  - Adding your own options (waiting for call screen) . 68
  - Options while waiting for a call 69
  - Options while a caller is online 71
  - Command overviews 72
    - From the main prompt 72
    - From the message base prompt 74
    - From the file SIG prompt 76

- o From the SysOp command processor . . . 77
- **o The off-line User Editor 79**
  - o Utilities 79
  - o Add User 80
  - o Print 80
  - o Edit User 81
- **o Understanding security levels for BBS Express! ST . . 82**
- **o Setting up your New User and Visitor Record . . . 85**
- **o Fine' 86**

Introduction

Welcome to BBS Express! ST

The importance of registering your product with us

If you call us with a technical question about BBS Express! ST, you must give us your serial number before you receive any technical help. The serial number will only be valid for 1 week after it is first used unless we receive your registration card or already have it on file. That is why it is very important that you send your product registration card to us.

This registration is only valid for the original purchaser of BBS Express ST! and is not transferable!

Our technical staff is available to answer questions Monday - Friday, 8 am - 5 pm CST, but please read the manual first before calling with questions.

This document and our technical support relies on the user having an understanding and knowledge of GEM, TOS, and proper operating procedures of your Atari ST computer.

If there is a file named READ.ME on the main program disk, it contains last minute information about BBS Express! ST which may not be covered in this manual.

Terms used throughout this manual

Pathname - The full pathname of a filename. For example, if you had a file named 'c:\userlog\userlog.dat', then the pathname would be 'c:\userlog\' (note the ending '\'!).

Filename - A fully qualified file name, as in 'c:\userlog\userlog.dat' (may include the Pathname).

Userlog - This is the 'User Database' file which contains the information on each user of your BBS. This file contains such things as each user's name, address, last call date, etc.

Call Log - A call log is where information on each of your calls gets written. For each user that calls, certain information such as the time of the call, the name of the user that called, etc. will be written to the call log file. The call log file can reside on disk, be written to a printer, or be turned off.

File SIG - A file SIG (Special Interest Group) is simply a 'file area' definition which allows you to segment your upload and download files. For example, Atari files could be divided into ST user areas and 8-bit user areas.

Help Files - Help files are pure text files which are used as menus to help your users understand your BBS. They are identified by the extension of **.HLP** (or **.COM/.SCR** if it is a compiled Script file). Help files can range from simple text files to be displayed 'as is' to extremely large and powerful MENU files or SCRIPT files.

Script Files - Script files are a special breed of Help Files. BBS Express! contains its own language known as the 'Script Language' which you can use to customize your BBS by writing your own commands. Scripts files are identified by their extensions of **.COM** or **.HLP**, depending upon whether the script file has been compiled off-line.

SYSDATA - The SYSDATA file is probably the most important file to the BBS program. This file, whose filename is always **SYSDATA.DAT**, contains such information as:

- o Which drive each message base resides on
- o Which drive each file SIG resides on
- o The baud rate capabilities of your modem
- o The command definitions for the BBS program
- o The pathnames of your help files.

===== - Is used as a border to enclose any text file displayed in this manual.

'x' - This refers to an alpha letter such as a,b,c,...etc. It is usually used to refer to drive (can be floppy, RAMDISK, drive partition, etc.).

'n' or 'nn' - This refers to a number such as 1,2,3,...etc. or 01,02,03...etc. (The number of required digits is shown by the number of n's.)

BBS Express! ST is not case sensitive. This means commands and menu selections may be entered in upper case, lower case, or any combination of cases.

Getting Started

Backing up your Master diskette

Before we go any further, you should back up your master diskette. The disk can be backed up from the GEM desktop by simply placing the master in drive A, a blank disk in drive B, and then 'dragging' the drive A icon onto the drive B icon.

We have chosen to not copy protect BBS Express! ST for your convenience. Please help us out by not giving or "loaning" a copy of the program to anyone. There are DEMOs available if anyone is interested in trying without buying. Software "rental" of this product is strictly illegal. Following these guidelines will protect your investment in BBS Express! ST and your computer by allowing us a reasonable profit and providing you with continued support. Without it, we both lose.

How Express! allocates its data files (Message bases and Userlog)

Express! pre-allocates all of its major datasets ahead of run time (or, we should say, you do). Why pre-allocate the datasets? Well, the main reason is speed. By having datasets pre-allocated to their maximum allowable size, Express! doesn't have to worry about either (1) adding onto the end of a dataset or (2) accidentally filling up a diskette.

Pre-allocation also has its benefits for you, too. You can feel secure in knowing that virtually no file size growth will occur on disks containing your message base and user database files. The only actions that can cause disk space to be used are a user uploading a file and callers being written to the call log dataset. This can be invaluable if you are running from floppy disks and/or RAM disks.

The pre-allocation technique will be covered later in the chapter 'Creating Your Userlog Database and Message Bases'.

Determining the setup best for your system

The highest occurrence of disk access under BBS Express! ST will be that of the help (.HLP) files. For this reason, it would be best to place these files in a RAM disk. On a 520ST with TOS in ROM, you should be able to allocate a RAMdisk of 100K (this is with no large .ACC files in your machine). With a 1040ST, you can safely allocate a ramdisk of approximately 600K.

The RAMdisk is also the perfect place to put the call log dataset. This will allow almost instantaneous writes to the file during the user's logon. Then, at your convenience, you can use the VIEW and PRINT commands from the SysOp command processor to see the call log. NOTE: If the call log fills up the destination disk, the logon procedure will become slow. No more caller data will be logged until more disk space becomes available. The previous callers' data will remain intact and valid.

All other disk files used (or produced) by Express! are volatile files. These should be placed on either a floppy drive or a hard drive. As we have said, a hard disk is highly recommended to get the most out of BBS Express!. If you don't have a hard drive, then your BBS is going to be severely cramped. If you are planning on running a floppy-only system, then it would be best to place your user database and message bases on drive A, and use drive B only for upload and download files.

Creating your User database and message bases

Loading the CREATOR program

The file CREATOR.PRG on your BBS disk is the program used to allocate both your user database and message bases. To load this program, place your program disk in drive A and double click on the CREATOR.PRG file.

Setting up your Userlog

First, we will create your user database file. This is the file that will hold the information on all of your BBS users. Move the mouse to the 'Create' on the menu bar. You will be presented with two options; click on the 'Create Userlog...' entry.

You will now be presented with a box that has an up and a down arrow. Next to the arrows you will see a 'Number of Users:' entry. The up and down arrows can be used to change the number of users that you would like to allocate. You will notice that you are shown, at the bottom of the box, the disk space required for the number of users currently selected.

Use the up and down arrows to select the maximum number of users that you want on your BBS system. When you are finished, click on the 'Create' box. Next, you will be asked which drive you want the user database file to be created on. Click on your selection.

After choosing the target drive, you will be presented with another box that will ask 'Create file as x:\USERLOG\USERLOG.DAT?' (where 'x' is the drive you just chose). Click on the YES box and your userlog will be created.

Setting up your Message Bases

Next, we must create each of your message bases. Move the mouse to the 'Create' option on the menu bar and click on the 'Create Message Base..' option.

You will be presented with a box that looks much like the one in the 'Create Userlog' option that we just performed. The main difference is that there are two sets of arrow keys; one controls the 'Number of messages' and the other controls the 'Bytes per message'.

The 'Number of messages' controls the maximum number of messages that can be present on this message base. This number can range from 5 to 250. The 'Bytes per message' controls the maximum size of each individual message. A user will not be able to save a message that is larger than the number that you enter here.

Click on the up and down arrows to choose the size parameters for this message base. The disk space required for each different allocation is again shown to you at the bottom of the box.

When you have finalized your size parameters, click on the 'Create' box. You will be presented with another box that asks two questions:

(1) Which message base are we creating?
(2) What is the name of this message base?

You must answer question number one with a number from 1-32. The 'public' message bases are numbered from 1-31, while the Electronic (private) Mail base is always number 32. You should ALWAYS create a #32 message base.

For question number two, you should enter in the name that you want associated with this particular message base. Some examples might be 'General Base' or 'Atari ST Users'. This is the name that the user will see on the command line when they are accessing this message base.

Creating the SYSDATA file for your system

What is the SYSDATA file?

The SYSDATA file is the most important file to the BBS. This file tells the BBS where all of your message bases, userlog, and menu files are. It also specifies the different parameters for your BBS such as the main command definitions, the baud rates your modem can access, whether you want feedback to go to the printer or to Electronic mail, etc.

Loading the SYSEDIT program

The file SYSEDIT.TOS on your data disk is the program that allows you to edit your SYSDATA file. To load and execute this file, simply double click on the SYSEDIT.TOS file from the GEM desktop, or enter 'SYSEDIT' <return> if you are using a command line interface.

Upon loading, the program will attempt to load a file called SYSDATA.DAT from the CURRENT active drive. If the program can not find this file, it will tell you that it is creating a new SYSDATA file.

What this means is that you should keep the SYSEDIT.TOS file on the same disk that your SYSDATA.DAT file is on. If you are running from a hard drive, then you will need to place the SYSEDIT.TOS file in the same folder as the SYSDATA file (which will normally be the root folder).

After the loading of the SYSDATA file, you will be presented with the main menu. This menu has 6 options on it, which are:

1. Edit the miscellaneous variables
2. Edit the message base definitions
3. Edit the file SIG areas
4. Edit the main command definitions
5. Edit the message base command definitions

X. Exit this program

At times, an '*' will appear next to some (or all) of the menu numbers. This '*' signifies that you have accessed this selection during this session with the SYSEDIT program.

The Miscellaneous Variables

The miscellaneous variables are made up of the following fields:

A. Hi message number
B. Total calls rec'd
C. Calls rec'd today
D. Secondary password
E. Min baud rate
F. Max baud rate
G. Starting baud rate
H. Feedback to email/printer
I. Userlog file name
J. Help 80 path name
K. Help 40 path name
L. Logfile name
M. Keep new users until edited?
N. Allow downloading past time limit?
O. Allow use of handles?

To change any of these fields, press the letter associated with it, and then key in the new information. Below is a detailed description of each these fields.

A. Hi Message Number

This number is used by the BBS whenever a user posts a new message. Each time a message is posted, this number will be increased by one. This allows the BBS to distinguish between messages that have or have not been read by a user. This is possible because each user's record also includes a 'last message read' field. This field should be set to one for your new BBS setup, and you should not change this field again once your BBS is up and running.

B. Total Calls Received

This field contains the total number of phone calls that your BBS has received in its lifetime. If you are currently running a BBS, you can place your 'total calls received' from your old system into this field; if not, then set this field to zero.

C. Calls Received Today

This field contains the number of calls that your BBS has received in today's 24 hour period (from 12:00am to 12:00am). The BBS program will reset this field to zero every day at midnight.

D. Secondary Password

The secondary password field is a safeguard against someone 'cracking' into your BBS with a sysop level access code (which means they could destroy anything they wished). Any user who gets past the initial logon sequence (that is, the number/password check) will be examined to see if they have SysOp

security levels. If they do, they will be prompted to enter in this secondary password. The user will only get one try at entering this password; a wrong entry will automatically hang up the phone.

E. Minimum Baud Rate

This field represents the minimum baud rate that you want your BBS to support. It DOES NOT mean the minimum baud rate that your MODEM can support! This makes it possible for you to, for example, accept only 1200 baud (or faster) callers.

F. Maximum Baud Rate

This field functions identically to the minimum baud rate field, except that it specifies the maximum baud rate that you want your BBS to support. Do not set this field to a value higher than what your modem can support!

G. Starting Baud Rate

Use this field to specify which baud rate you want the BBS to first answer the phone at. This determines the baud rate that the initial 'Press <ASCII> Return...' will be sent at. BBS Express! looks to receive a valid 'CR' first at this baud rate. If any other character is received, it switches the MODEM to the next consecutive higher baud rate. This continues until the maximum baud rate is reached. After that the MODEM will be switched to the minimum baud rate. Of course if a valid 'CR' is received at any time, the BBS connects at that baud rate.

H. Feedback to Email or Printer

By changing this field, you tell the BBS what you would like to do with 'Feedback to SysOp' messages that users leave. If you specify 'To Email', then any feedback messages will be placed in the Electronic Mail base, just as if the user had used the command to 'S'end Mail to you. This is usually the preferred way; it's much faster than logging the messages to the printer. Another advantage is that you can easily read and answer any messages to you by using the reply command.

I. Userlog File Name

In this field, you must specify the FULL filename of your USERLOG.DAT file that you created earlier with the CREATOR program. Some examples might be 'c:\userlog\userlog.dat' or 'd:\bbs\userdata\userlog.dat'.

J. Help 80 Pathname

This is a required field in which you specify the path name of the 80-column menu folder.

K. Help 40 Pathname

This is an optional field in which you specify the path name of the 40-column menu folder.

The menu (.HLP) files are accessed as follows: if the user is in 80-column mode, then the BBS will always display the help files from the 80-column path specified above. If the user is in 40-column mode AND you have specified a 40-column path above, then the BBS will first search for the help file in the 40-column path you have specified here. If the file is not found in the 40-column path, then the BBS will return to the 80-column pathname to look for the file.

Menu file searches are done in the following order: (1) the path is searched for a .HLP file, (2) if no .HLP file is found, then the program searches for a .COM file (a COMpiled script file), and (3) if no .COM file is found, then the program will search for a .SCR (SCRipt) file.

L. Logfile Name

This is an optional field in which you specify the full filename of the caller log. This file, if specified, contains information on each of the callers that your BBS has received. It contains such data as:

- Who the caller was
- Where he/she called from
- What time they logged on/off
- Which files they up/down loaded
- Any 'System Errors' that have occurred
- Any 'Script file' errors that have occurred

You can specify three different types of entries here:

1. A valid filename.
2. The word PRINTER will send the log file to your printer.
3. Nothing. In this case, no call log will be written.

The preferred place to write the call log is to a RAM disk. This has many advantages; here are the main ones:

- It is extremely fast; much faster than writing to a printer.

- It is convenient; you can print or view the call log at any time with the VIEW or PRINT commands from the SysOp command processc

M. Keep New Users Until Edited?

There is a flag present in each user's data record that signifies wheth or not this user is 'new'. To the BBS, a new user is one that has logged to your system, but you have not yet edited their record (either with the n-line or off-line user editor). If this field is set to YES, then the new user flag will remain set in a user's record until you edit their record. If NO is specified, then the new user flag will remain set until they log on for the second time; at that point, the flag will be cleared.

E. Allow Downloading Past Time Limit?

If this flag is set to NO, then the BBS will not allow a user to download a file if the time required to do so would put them over their time limit. As an example, let's assume that a user is on-line who has a 30 minute time limit. This user has been on for 15 minutes this call; they now have 15 minutes remaining. If they attempted to download a file that would take 20 minutes to download, the BBS would not allow the download to take place.

O. Allow use of Handles?

If this flag is set to NO, then NEW users will not be allowed to enter a handle. The handle will still be in use, but it will simply be set to the 'Real Name' that they enter in (ie: the name and handle will be identical).

The Message Base Definitions

There are three purposes behind the message base definitions. These are to tell the BBS:

1. how many message bases your system has
2. the path name containing your message bases
3. the drive that each message base resides on

First, press 'A' and key in the number of message bases that your system is going to have. Note that you DO NOT count the Electronic Mail base; only those bases numbered from 1 - 31.

Next, press 'B' to specify the path name (minus the drive specifier) that your message bases will reside in. This means that if you have your message bases spread across two or more disk drives, then each drive must have this path name on it. NOTE: Although you may have your message bases use more than one drive, a single message base may not spread across two drives.

Finally, press 'C' and tell the BBS which disk drive each message base resides on. Valid entries are 'A' - 'P' and blank. The message bases 1-31 are numbered consecutively so you cannot leave blanks in between bases (except before 32).

Edit The File SIG Areas

You have at your disposal 32 'file areas' (also called SIGs) available for uploading and downloading activities. These different areas allow you to break your files up into sections that contain related files.

As an example, here is one way that you might segment your file areas:

1. Atari ST Games
2. Atari ST Utilities
3. Atari ST Applications
4. Atari ST Communications
5. Atari 8-bit Games
6. Atari 8-bit Utilities
7. Atari 8-bit Applications
8. Atari 8-bit Communications
9. Other Files

When you enter the 'Edit File Area' screen, you will be presented with a screen showing the 32 files areas (many which will say '(not in use)'). To add a new (or edit an existing) entry, simply press 'E' followed by the number of the SIG to edit. After entering the number, you will be presented with another screen with the following information on it:

A. the name of this SIG
B. the pathname of this SIG
C. whether or not to use descriptions on this SIG
D. whether or not to show free space before uploads to this SIG

First, press 'A' and enter the name of this SIG. This is a string such as 'Atari ST Games' or 'Atari 8-bit Communications'. This is the name that the user will see when selecting a file area to enter.

Next, press 'B' and enter in the pathname that this SIG will occupy. Some examples are 'c:\bbs\st_games' or 'a:\utils\'. You will have to create this folder before running your BBS program.

Next, press 'C' to toggle the 'Use descriptions on this SIG?' field between Yes or No. If you specify 'Yes', then you must create a folder named DESC within the folder you specified in option B above. In other words, if you specified a pathname of 'c:\bbs\st_games\' in option B above, then you would also have to create a folder called 'c:\bbs\st_games\desc\'. This is where the 500-byte descriptions for each file will reside.

Finally, press 'D' to toggle the 'Show Free space?' field between Yes or No. If you specify Yes, then the on-line user will be shown the free disk space available before they begin an upload. If you have a hard drive, you should specify 'No' here, since getting the free space of a hard drive is very slow (sometimes taking more than 10 seconds).

Edit the Main Command Definitions

The commands for BBS Express! are specified in a command table which you can modify with this screen. Each table entry is broken down into the following specifications:

User Key - the key the user hits
BBS Key - the key the BBS understands
Sec Lvl - the security level needed to execute this command
Display - the string to display when the user presses this key
File to View - the .HLP file to view (if it is not a BBS command)

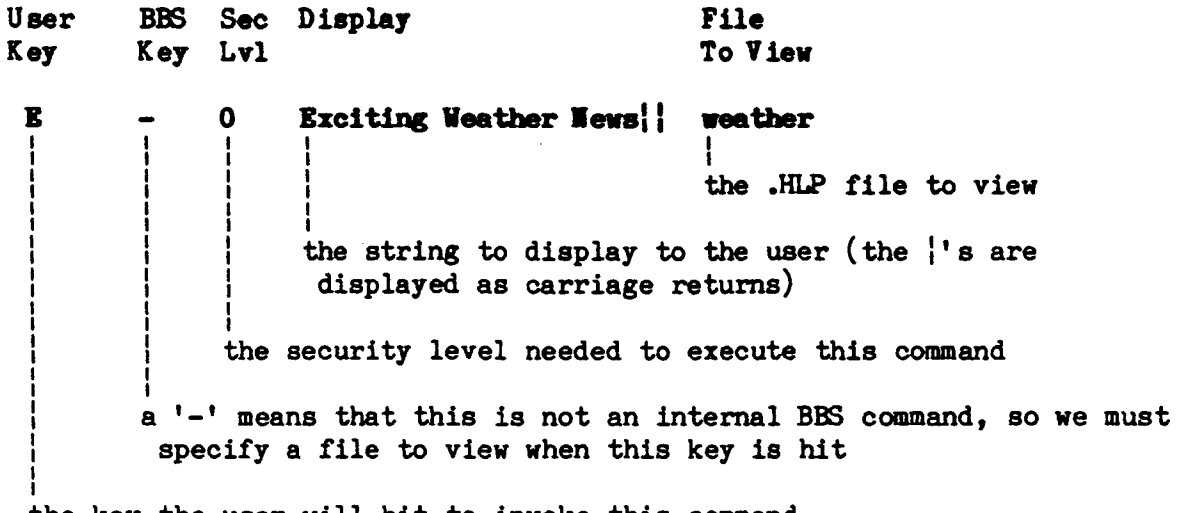
As an example, let's say that you would like to change the 'C' command (Call for SysOp) to the letter 'A' (Awaken the SysOp). This could easily be done by changing the entry to the following:

```
User    BBS  Sec  Display                          File
Key     Key  Lvl                                   To View

 A      C    0    Awaken the SysOp¦¦

 ¦      ¦    ¦    ¦
 ¦      ¦    ¦    ¦
 ¦      ¦    ¦    ¦
 ¦      ¦    ¦    the string to display to the user (the ¦'s are displayed
 ¦      ¦    ¦     as carriage returns)
 ¦      ¦    ¦
 ¦      ¦    the security level needed to execute this command
 ¦      ¦
 ¦      ¦
 ¦      the key the BBS understands as 'call for the sysop'
 ¦
 the key the user will hit
```

While editing the table, an entry of '-' (a dash) represents a 'null' entry.As another example, let's assume that you would like to add a brand new command to your main menu commands for users to read the daily weather report. First, you must decide on a letter that is not currently being used as a main command. In our example, we'll use the letter 'E' and call the command 'Exciting Weather News'. Here is the command entry that would install this new command:


User Key
BBS Key
Sec Lvl
Display
File To View
E
-
0
Exciting Weather News¦¦
weather
the .HLP file to view
the string to display to the user (the ¦'s are displayed as carriage returns)
the security level needed to execute this command
a '-' means that this is not an internal BBS command, so we must specify a file to view when this key is hit
the key the user will hit to invoke this command

As you can see from above, entering a dash in the BBS Key field tells the BBS that this is not one of its internal commands. When the BBS sees this, it will display the help file named under the 'File To View' column. This file must be present in your 40 or 80 column help file pathname, under the name WEATHER.HLP. This is the same technique that you use to invoke your menu files or script files.

There are two pages of these commands; you can toggle between the pages by pressing the <return> key. On the second page, there is one additional entry named 'Display on Invalid Command'. This field specifies the string that will be displayed when the user presses an unknown key from the main command prompt.

Edit the Message Base Command Definitions

Editing the message base command definitions is identical to editing the main command definitions, except you are specifying the commands that are available from the message base command prompts. There are probably few times that you will need to change any of these commands.

One command that you should NEVER change is the 'X' (exit) command. This is because the BBS will force an 'X' key from the user if they exceed their time limit while in a message base.

The Help File Menus

What are help files?

In the context of BBS Express!, a 'help file' is a text file (normally with a name such as MAINMENU.HLP) that can be viewed by a user, primarily for help in using your BBS system. In a much broader scope, help files can also be MENU files or SCRIPT files (and even compiled script files!).

Help files can give special instructions to the BBS program through special keywords on the first line of the file (and ONLY the first line of the file). These keywords can exist anywhere on the first line, and are listed below:

MENU - tells the BBS that this is a menu file. This means that the user must input some data to select a specific file to view.

SCRIPT - tells the BBS that this is a script file (explained later).

NOABORT - tells the BBS that the user cannot abort out of viewing this file (with the ^X or ^C key).

RETURN - used in conjunction with a MENU file. This instructs the BBS to return to this menu file after viewing any text file that the user selects.

SURVEY - tells the BBS that this is a User Survey file. More on this later.

Help files can also access certain BBS variables for display purposes. This is accomplished through what are called the 'ampersand variables'. When displaying a text file to the user, the BBS will look for an '&nn', where 'nn' is a TWO-digit number. If one is encountered, then the corresponding variable data will be displayed instead of the '&nn'. If you should ever need an ampersand to appear in one of your help files, simply place two of them together, as in:

```
This && That
```

which will display to the user as:

```
This & That
```

Here is a list of the variables available to you, along with the functions that they perform.

&00 -clears the screen
&01 -displays the user's Handle
&02 -displays the user's Real Name
&03 -displays the user's Password
&04 -displays the user's Address
&05 -displays the user's City
&06 -displays the user's State
&07 -displays the user's Zip Code
&08 -displays the user's Phone Number
&09 -displays the user's Age
&10 -displays the user's Computer Type
&11 -displays the user's Video Width (40 or 80 columns)
&12 -displays the user's Last Call Date
&13 -displays the user's Time Limit per Call
&14 -displays the user's Time Limit per Day
&15 -displays the user's Minutes on so far today
&16 -displays the user's Last Read Message Number
&17 -displays the user's Download to Upload Ratio
&18 -displays the user's number of Downloads
&19 -displays the user's number of Uploads
&20 -displays the user's number of Messages Posted
&21 -displays the user's number of E-mails Sent
&22 -displays the user's Total calls
&23 -displays the Current Time
&24 -displays the Current Date
&25 -displays the Current System HiMsg number
&26 -displays the calls the BBS System has received
&27 -displays the Total calls received today
&28 -displays the Last Caller
&29 -displays the Current Baud Rate of the caller
&30 -displays the user's Country
&31 -displays the current user's User Number.
&32 -displays the minutes the user has been connected (excluding chat and upload time)
&33 -displays the user's logical time limit this call
&34 -displays the current time in military time (always 8 characters in the form of 01:14:24 or 19:34:20)

&50 -sets the Foreground to White
&51 -sets the Foreground to Red
&52 -sets the Foreground to Green
&53 -sets the Foreground to Black
&54 -sets the Background to White
&55 -sets the Background to Red
&56 -sets the Background to Green
&57 -sets the Background to Black
&58 -turns reverse-video (inverse) text On
&59 -turns reverse-video (inverse) text Off

In its simplest form, a help file is a pure text file that contains no special instructions for the BBS program. One example of this is the file that is displayed when a user logs off of the system, LOGOFF.HLP:

```
Thanks for calling, &01.

We hope you enjoyed calling our BBS!

Call Back Real Soon!
```

The Menu Option for Getting User Input

Pure text files are fine, but what if you have a library full of text files? You need some way to present a menu to the user, allowing him/her to select the file they wish to view. That is where the MENU option comes in.

The word '**MENU**' appearing on the first line of a help file tells the BBS that special processing must take place. Also, certain stringent guidelines must be followed in setting up this menu file. The easiest way to explain MENU files is with an example:

```
menu
Enter 1-2, <RETURN> To Exit:
3
1¦a:menus_80\exp1030.doc
2¦a:menus_80\hints.doc
@¦exit
&00
```

BBS Express! ST
Copyright (c) 1987 ICD, Inc.

The Library

1. 1030 Express! Documentation

2. Hints on using The Flash Terminal Program

Looking at the preceding example, here is the syntax required for menu files:

```
line 1      : must contain the word 'MENU'
line 2      : is the prompt you want the user to see
line 3      : contains the number of selections available
```

lines 4-nn : contain the selection criteria available to the user. The number entered in line 3 corresponds to the number of selection criteria lines that should be present.These entries take the following form:

<user input>|<file to view>

An '@' sign in the <user input> field means that the user pressed **only** the **RETURN key.**

Also, a <file to view> entry of '**EXIT**' tells the BBS to just exit out of this menu file without a selection being made.

rest of lines: is the menu file displayed to the user prior to prompting them with the string specified in line 2.

User input is not limited to 1-character input; you may use whole words if you wish. Below is another example of a menu file that uses whole words for its selection criteria:

```
menu return
Please enter your selection (or 'quit' to abort):
4
may|c:\news\may.txt
june|c:\news\june.txt
july|c:\news\july.txt
quit|exit

&00

                    Atari Club News

          Which newsletter would you like to view?

                    May, June, or July?
```

In this example, the user would be shown the menu, and then prompted with the string specified in line 2. Then, depending on whether they entered in 'May', 'June', or 'July', the file may.txt, june.txt, or july.txt would be shown to them. If the user were to enter in 'quit', then execution of the menu file would cease immediately.

You will also notice that the keyword 'RETURN' is used in the first line of this menu. This means that after the user finished reading one of the month's newsletter files, they would be returned to this menu, once again getting the above menu file displayed to them.

The SURVEY Option for Taking User Surveys

There are times when you will want to poll your BBS users for their input on certain topics. This can be accomplished by the 'Survey Files' that BBS Express! allows you to create. The BBS will only allow a user to take a survey one time; on subsequent tries they will merely be shown the current results of the survey. BBS Express! allows you to have up to 32 surveys going at one time.

Survey files are much like MENU files in their layout. The main difference is that the results of the survey will be placed in the <file to view> area of the selection criteria.

As an example, let's assume that we are thinking about adding a new message base and a new file SIG area to our BBS system. What we would like to find out from our users is which of the choices we have picked they would prefer to see implemented.

Here are the survey files that we could use to accomplish this:

File: c:quest_1.sur

```
survey 1
Please Enter Your Selection:
4
1¦00000
2¦00000
3¦00000
4¦00000
c:quest_2.sur

&00
```

Message Base Survey

We have narrowed our search for a new message base subject down to the following areas.

Which of them would you prefer to see implemented?

1. Programming
2. World Affairs
3. Politics
4. Dear Abby

File: c:quest_2.sur

```
survey 1
Please Enter Your Selection:
3
1¦00000
2¦00000
3¦00000
exit

&00
```

File Area Survey

We are also looking for a new topic for a file area. Which of these below would you prefer?

1. Programming Examples
2. 8-bit Utilities
3. ST Utilities

Looking at the preceding examples, here is the syntax required for survey files:

line 1 : must contain the word '**SURVEY**', along with the number (1-32) of this survey. This number is used to mark that a user has already responded to this survey.

line 2 : is the prompt you want the user to see

line 3 : contains the number of selections available

lines 4-nn : contain the selection criteria available to the user. The number entered in line 3 corresponds to the number of selection criteria lines that should be present. These entries take the following form:

<user input>¦<number of times this has been selected>

the next line : the line immediately following the selection criteria contains the filename of the next survey file (question) to display. This allows each survey to have an unlimited number of questions. The word '**EXIT**' in the second survey file means that this is the last question in this survey.

rest of lines : is the menu file displayed to the prior to prompting them with the string specified in line 2.

Again, user input is not limited to 1-character input; you may use whole words if you wish.

The most important thing to remember about Survey Files is that the numbers are actually updated **IN THE SURVEY FILE ITSELF**. For this reason, you must place five zeros after the <user input> specification; any other number of zeros will cause the survey file to be partially overwritten!

Also, you should be very careful if you place the survey files in RAM disk. Since the results are contained in the file, some result data could be lost if you turned off your machine without copying the survey files back to a floppy or hard disk.

Since we highly recommend the use of a RAM disk for all of your menu files, here is the simplest solution. Keep your survey files on floppy/hard disk, and invoke them from a menu file which can be kept in RAM disk (yes, SURVEY files can be invoked by selection from a MENU file). That way, you never have to worry about losing survey results because of a power failure.

The Script Language

Introduction

BBS Express! has its own programming language known as the **Script Language** (also called simply script files). Script files allow you much more control than menu files do; but with this control comes more responsibilities on your part. The Script Language in BBS Express! ST is theoretically no different than writing in BASIC, C, Pascal, etc. It is a programming language that allows you to write your own 'commands' for the BBS while still under the watchful eye of the BBS program itself.

The Script language processor is a mix of both an **interpretive** language and a **compiled** language in that it interprets each command as it is encountered, yet it is **very** fast because it is compiled (really 'tokenized') ahead of run-time so that the interpreter has very little work to do.

This 'compiling' can occur in two ways. First, you have the ability to compile your script files off-line with the **SC.TTP** program. Alternately, you can let the script files be compiled on-line at the time they are invoked. This on-line compilation occurs very quickly; an 8,000 byte script file will be compiled in just under two seconds. The BBS program knows if a script file is in its compiled format as soon as it is loaded in; if it has not been compiled off-line, it will compile the file at load time. **NOTE:** If a script file has not been compiled off-line with SC.TTP, it will be compiled on-line **every** time it is executed.

One advantage of compiling scripts off-line is that the script files can be much larger (up to 32,000 bytes in length) in un-tokenized form. When the BBS loads a script file, its size limit is 16,000 bytes. But, if the file has been compiled off-line, then this is 16,000 bytes in **tokenized** form. So, it is possible to write some huge script files to perform very complex functions.

The 16,000 byte limitation exists because the entire script file is loaded into memory at one time (so it can be executed at a much faster pace). This size limitation is not a big issue, since you can easily 'chain' to another script file through the use of the EXECUTE command. Not only that, but a 16,000 byte tokenized script file is a **huge** program. If scripts are pre-compiled off-line with the script compiler (**SC.TTP**), there should never be a need to chain to another script file.

When writing a script file, you have 100 variables at your disposal, which you define with the **DEFINE** statement (see below). These variables are all actually strings, but numeric calculations and comparisons are possible because the script file processor will convert numeric strings to their integer equivalents before executing a command.

In addition to the 100 variables available, there is also a **table** of 100 entries available for your use through the **TABLE_xxx** commands. This allows high speed loading and manipulation of data tables.

There are some very strict rules on syntax of the statements. Let's go over these rules first:

- **The system is very flexible as far as 'free-forming' the code. It doesn't matter if code starts in column 1 or column 20. This is a big advantage since indenting makes the 'top-to-bottom' code much easier to read. The only exception to this rule is label names. Label names MUST begin in column one!**

- Comments are preceded by a ';' (semi-colon), and can occur anywhere on a line. If you are writing a large script file, please use comments! The off-line script compiler removes all comments when tokenizing a script file, so comments do not effect your 16,000 byte limit (note that this is only the case if the script is compiled off-line; if compiled at run-time, then comments do eat up bytes out of the 16,000 byte limit).

- The system is not case sensitive. Everything is folded to upper case before execution, so you can say

```
     if option = 'quit' then
or   IF Option = 'QUIT' THEN
or   If OPTION = 'QuIt' Then
```

  all giving the same results.

- There can only be ONE statement per line. All conditions under an IF statement MUST be on the next line -- anything after the 'condition check' is simply ignored.

- Variable names enclosed within quotes (as in a print statement) **must** be enclosed within brackets ([]). For example, the following line prints out the variables FIRST_NAME and LAST_NAME in one print statement:

```
printe 'Your name is [FIRST_NAME] [LAST_NAME]'
```

- All string constants must be enclosed in single quotes.

- IF statements MAY NOT be nested.

- You can also access the 34 'user variables' just like you do in your MENU files. These are still known as &1 - &34 (notice that you don't have to pad a 0 in front like you do in the MENU files). As an example:

```
SET user_name = &1
```

  will set the work variable USER_NAME to the user's handle.

- Labels must end with the ':' character, and must be the only thing on the line (and, they **must** begin in column one).

- If you do an EXECUTE command to chain to another script file, all of the work variables will remain intact.

- A GOTO statement automatically clears out any IF statement processing that you were doing. So, it's perfectly alright to GOTO a label outside of an IF..ENDIF block.

- A script file can be invoked from a MENU file, but a MENU file **CANNOT** be invoked from a script file.

- All parenthesis are ignored, so they may be used to enclose any statements that you prefer (it's simply a matter of personal taste). Also, either commas or spaces can be used to terminate a word. To illustrate, the **PrintE** command could be written in any of the following ways:

  ```
  PrintE 'Greetings, how is your day going?' center

  PrintE ('Greetings, how is your day going?') center

  PrintE ('Greetings, how is your day going?', center)

  PrintE 'Greetings, how is your day going?' (center)
  ```

Command Overview

We will now go over the statements available to you while writing a script file. In the command listing that follows, the following symbols will be used:

UPPER CASE - Those words in upper case are **keywords.**

¦ - The '¦' symbol stands for '**or**'. The line

SCREEN ¦ TABLE

should be read as 'the keyword screen **OR** table'.

[..] - Brackets enclose any **optional** keywords that you may include if you wish.

<var> - means that the name of one of your DEFINED work variables should be placed here.

'string'- means that a text string should be placed here. This may be either a string of letters enclosed within single quotes, or one of your DEFINED variable names.

&nn - means that one of the 34 **user variables** should be placed here, in the form of &1, &2...&34.

Page #	Alphabetical Script Language Command List
42	**ABORT** OFF ¦ **EXIT** ¦ label name
53	<target_var> = **ARRAY_GET** (<source_var>, index_into_array)
53	**ARRAY_PUT** (<var>, index_into_array, 'character')
54	<var1> = **ARRAY_FIND** (<var2>, 'character')
53	**ARRAY_CLEAR** (<var>)
57	**BACKGROUND** (WHITE ¦ RED ¦ GREEN ¦ BLACK)
31	**BACKSPACE** (number)
31	**BELL**
45	**BREAK**
46	**CLEAR** (SCREEN ¦ TABLE)
32	**CLOSE**
46	<dest_var> = **COPY** (<source_var>, start_pos, length)
46	**DECREASE** <var> [amount]
28	**DEFINE** ¦ **VAR** ¦ **INTEGER** ¦ **STRING** variable_name [= 'initial value']
47	**DISCONNECT**
31	**DUMP**
40	**ELSE**
41	**ELSEIF** ¦ **EF**
31	**END**
45	**ENDFOR**
41	**ENDIF**
43	**EXECUTE** ('filename.ext')
43	**EXIT**
47	**EXPAND** (<var>, length)
35	<var> = **FILE_COPY** ('source', 'destination')
35	<var> = **FILE_DELETE** ('filename.ext')
36	<var> = **FILE_PRINT** ('filename.ext')
36	<var> = **FILE_RENAME** ('oldname.ext', 'newname.ext')
36	<var> = **FILE_CHGDIR** ('folder')
33	**FINPUT** <var> [number_of_lines]
45	**FOR** <var> = start_value TO end_value [DO]
57	**FOREGROUND** (WHITE ¦ RED ¦ GREEN ¦ BLACK)
33	**FPRINT** 'string'
33	**FPRINTE** 'string'
37	**GEM** ('program.ext', 'command_line_parameters')
30	<var> = **GET_KEY**
58	<var> = **GET_WORD** ('string', word_number)
44	**GOSUB** ¦ **PERFORM** label
43	**GOTO** label
38	**IF** expression = ¦ < ¦ > ¦ # expression [THEN]
38	**IF COMMAND_LEVEL** (number) [THEN]
39	**IF MESSAGE_LEVEL** (number) [THEN]
39	**IF FILE_LEVEL** (number) [THEN]
39	**IF TIME_EXCEEDED** [THEN]
40	**IF FILE_EXISTS** ('filename.ext') [THEN]
40	**IF** <var> = **NUMERIC** [THEN]
46	**INCREASE** <var> [amount]
30	**INPUT** <var> max_length [NORETURN]
57	**INVERSE** (ON ¦ OFF)
48	<var1> = **LENGTH** (<var2>)

Page # | **Alphabetical Script Language Command List**

48 MARGIN (size)
34 <var> = NOTE
32 OPEN ('filename.ext' | PRINTER, INPUT | OUTPUT | APPEND)
34 POINT byte_offset_into_file
58 <var> = POS ('substring', 'string')
29 PRINT 'string' [CENTER]
30 PRINTE ['string'] [CENTER]
48 <var> = RANDOM (upper_limit)
44 RETURN
49 [SET] <var> = 'string' | %nn | &nn [+-*/] [<var>]
50 SHUFFLE <var1> <var2>
55 <target_var> = TABLE_GET (index_into_table)
56 TABLE_PUT (index_into_table, 'string')
56 <var> = TABLE_LOAD ('filename.ext')
56 <var> = TABLE_FIND ('string')
50 TIME_LIMIT (new_time_limit_in_minutes)
37 TOS ('program.ext', 'command_line_parameters')
58 <var> = TOTAL_WORDS ('string')
51 <var1> = VALUE (<var2>)
51 <var> = VERSION
34 VIEW ('filename.ext' [NOABORT])

Defining your variables

Syntax: **DEFINE 'name' [= 'initial value']**

Example: define users_name = 'Joe E. Smith'

Aliases: **STRING, ARRAY, INTEGER, VAR**

Usage: The define command allows you to specify the variables that will be used in this script file. All of your variables **MUST** be defined before being used!

Optionally, you may also define an initial value to this work variable in the DEFINE statement. Doing the following:

```
define  bbs_name  =  'BBS Express! ST'
```

is identical to doing:

```
define  bbs_name
set bbs_name = 'BBS Express! ST'
```

In addition, the words **INTEGER, STRING, ARRAY,** or **VAR** may be used in place of the word **DEFINE.** This makes absolutely no difference in the definition of the variable; it is merely allowed to suit your preference and to attempt to self-document. For example, if you had a field named NAME which is always a string, and a field named COUNT that is always used in a numeric fashion, it is probably better (for readability only, of course) to do:

```
string  NAME  =  'Joe Smith'
integer COUNT =  0
```

Console I/O

The following commands allow you to get input from and display output to the screen (both the sysop's screen and the current on-line caller's screen).

Syntax: **PRINT 'string' [CENTER]**

Example: print 'Welcome, &1, to BBS Express! ST'

Aliases: **WRITE**

Usage: Prints the string (with symbolic variable substitution allowed) without a carriage return at the end. If the optional keyword CENTER is specified, the line will be centered on the user's screen, dependent upon their current terminal width.

NOTE: If you want to print one of your variables in the string, it must be enclosed in brackets ([]).

Special characters may be printed in PRINT strings by prefixingthem with the backslash ('\') character. These character combinations are as follows:

\\ - prints a '\' character
\" - prints a sinqle quote (')
\{ - prints a '[' character
\} - prints a ']' character
\n - prints a carriage return/linefeed
\i - turns inverse video on
\o - turns inverse video off
\b - turns the foreground black
\r - turns the foreground red
\g - turns the foreground green
\v - prints a logical carriage return. That is, if the user is in 40 column mode, then a carriage return will be printed. If the user is in 80 column mode, a space will be printed.

For example,

```
printe '\{\rA\b\}  Option number A'
```

will print

[A] Option number A

with the letter 'A' in RED, and the rest of the line in black.

Syntax: **PRINTE ['string'] [CENTER]**

Example: printe 'Welcome, &1, to BBS Express! ST'

Aliases: **WRITELN**

Usage: Identical to the PRINT, except an EOL (carriage return and linefeed) is printed after the string. If no 'string' is specified, then only an EOL is printed.

Syntax: **INPUT <var> max_length [NORETURN]**

or: **<var> = INPUT (max_length, [NORETURN])**

Example: input user_input 14 noreturn

Usage: This is your tool for getting input from the user. You simply specify which of your work variables to read the input into, along with the maximum length allowed. If NORETURN is specified, then the user will not be able to hit just the <return> key. In our example, the user will enter a string, up to 14 characters long, into the work variable USER_INPUT. They will not be allowed to press only the <return> key.

Syntax: **GET_KEY <var>**

or: **<var> = GET_KEY**

Example: get_key user_input

Usage: The get_key command gets a single input key from the user. Note that get_key **does not** echo the character back to the user. This gives you the ability to capture the input key and then display a string to the user, as in:

```
y_or_n:
  printe 'Continue (y/n)? '
  user_input = get_key
  if user_input = 'y' then
     printe 'You bet!'
  else
     printe 'No way!'
  endif
```

Syntax: **BACKSPACE (number)**

Example: backspace (8)

Usage: This command allows you to backspace over previously PRINTed text. This command is intended to be mostly used when you are creating script files that do a lot of file I/O. Sometimes, if the disk I/O may be lengthy, you will want to print a line to the screen so that the user knows that something is happening. The BACKSPACE command then allows you to erase over that string so that your display doesn't look cluttered.

For example, let's say we are searching an AREA CODE phone list file to display to the user:

```
script
define area_code
define the_number

open ('e:\areacode.dat', input)
main:
    print 'Searching...'
    gosub find_area_code        ;go look it up
    backspace (12)              ;erase the searching...
    gosub display_line          ;and show the line
    goto main
```

Syntax: **BELL**

Example: bell

Usage: Prints a BELL character to the screen.

Syntax: **DUMP**

Example: dump

Usage: The DUMP command tells the processor to simply 'show what follows as textual information until you find an END statement'. This is useful if you are formatting a menu-type display and you don't want to be doing a lot of PRINTE statements. This is simply a raw dump of the lines that follow the DUMP statement. You CANNOT use the &nn variables if you are DUMPing lines (its one major drawback).

Syntax: **END**

Example: end

Usage: The END command signals the end of a DUMP command.

File Input/Output

The following commands allow you to do input and output functions from and to a disk file (or, possibly a printer).

Syntax: **OPEN ('filename.ext' | PRINTER, INPUT | OUTPUT | APPEND)**

Example: open ('c:\bbs\data\orders.txt', input)

Usage: The OPEN command opens the named file and readies it to be read from or written to. If the filename is specified as PRINTER, then the input/output/append mode specification is ignored. The following rules apply to opening files:

- o Only ONE file may be open at a time.
- o Input mode readies the file for READ processing only.
- o Output mode prepares the file for WRITE processing from the beginning of the file.
- o Append mode prepares the file for WRITE processing from the END of the current file; data is appended onto the end of whatever data is already there.
- o If a file is opened for INPUT that doesn't exist, the script file will terminate immediately with an error (you can avoid this by using the IF FILE_EXISTS command).
- o If a file is opened for OUTPUT that already exists, it will be **DELETED**! Be careful!
- o If a file is opened for APPEND mode that doesn't exist, it will be created.

Syntax: **CLOSE**

Example: close

Usage: Closes out the file currently open. No error occurs if a file is not currently open.

Syntax: **FPRINT 'string'**
FPRINTE 'string'

Example: fprinte 'Current caller is: &1.'

Usage: Writes the specified string to the file currently OPEN. Both FPRINT and FPRINTE will put an EOL at the end of the string, so it doesn't matter which one you use. If no file is currently open in write or append mode, the script file will be terminated immediately with an error. For example, let's say that you wanted to create your own 'call log' file. You could do this by tapping into the SYSNEWS file or the WELCOME screen:

```
script
open ('e:\log\my_log.dat', append)
fprinte                                    ;blank line
fprinte 'User Name: &1'                    ;log user's name
fprinte '     From: &5, &6'                ;log user's city/state
fprinte '    On At: &23 on &24'            ;log date and time
close logfile                              ;close the logfile
view ('e:\menus\welcome1.hlp')             ;view OLD textfile
exit
```

Syntax: **FINPUT <var> [number_of_lines]**

or: **<var> = FINPUT ([number_of_lines])**

Example: finput input_line

Usage: Reads a string from the file currently open for input into the variable specified. The string can have a maximum length of 80 characters.

If the optional **NUMBER_OF_LINES** parameter is specified, then number_of_lines-1 lines will be SKIPPED when reading the file.

For example:

```
finput input_line 6
```

would cause five lines to be skipped, and the sixth line would be read into the variable INPUT_LINE.

If an End-Of-File condition occurs on the read, the script file processor will immediately do a 'GOTO EOF' statement. So, anytime you are reading data in from a file, you should have a label named EOF: somewhere in your script file.

Let's say we have a product file somewhere that we just want to display to the user:

```
script
define product
open ('e:\products.dat', input) ;open the input file
read_em:
   finput product               ;read a line from the file
   printe 'Product: [product]'
   goto read_em                 ;and keep looping

eof:                            ;we'll come here on EOF
   close                        ;close out the file
   printe 'End of products'
   exit
```

Syntax: **NOTE <var>**

or: **<var> = NOTE**

Example: file_pointer = note

Usage: The NOTE command will return the current file position into the variable specified. This can be used later by a **POINT** command to return to the same position in the file.

Syntax: **POINT byte_offset_into_file**

Example: point file_pointer

Usage: The POINT command will set the file pointer to the position specified in this call. This value is a byte offset from the **beginning of the file.**

Syntax: **VIEW ('filename.ext', [NOABORT])**

Example: view ('c:\bbs\data\orders.txt')

Usage: The VIEW command can be used to view pure text files. The file will be displayed just as if the file were selected from a MENU file. At the completion of the viewing of the file, control will return to the statement following the VIEW statement. If you specify the optional third parameter NOABORT, then the user will not be able to ^X out of the viewing of the file. **NOTE:** The NOABORT option has no bearing on any ABORT option that you may have previously set.

Syntax: **<var> = FILE_COPY ('source', 'destination')**

Example: rc = file_copy ('c:\bbs\base_1.dat', 'e:\base_1.dat')

Usage: The FILE_COPY command will copy the source file to the destination file. The result of the copy function will be returned into the variable specified. The result will be a zero if the operation completed successfully, and a 1 if the operation failed for any reason. **Wildcard copies are not allowed with the file_copy command!** An example:

```
script

integer rc

rc = file_copy ('e:call.log', 'c:\data\call.log')
if rc > 0 then
   printe 'Unable to back up call log!'
endif
```

Syntax: **<var> = FILE_DELETE ('filename.ext')**

Example: rc = file_delete ('e:\call.log')

Usage: The FILE_DELETE command will delete the file specified. The result of the delete function will be returned into the variable specified. The result will be a zero if the operation completed successfully, and a 1 if the operation failed for any reason. **Be extremely careful with this command!** To build on our previous example:

```
script

integer rc

rc = file_copy ('e:call.log', 'c:\data\call.log')
if rc > 0 then
   printe 'Unable to back up call log!'
   printe 'Call log dataset retained!'
   EXIT
else
   printe 'Call log successfully copied...'
Endif

rc = file_delete ('e:call.log')
if rc > 0 then
   printe 'Unable to delete call log dataset!'
else
   printe 'Call log successfully deleted.'
endif
```

Syntax: **<var> = FILE_PRINT ('filename.ext')**

Example: rc = file_print ('e:\call.log')

Usage: The FILE_PRINT command allows you to print a file. Again, the result of the print function will be returned into the variable specified. The result will be a zero if the operation completed successfully, and a 1 if the operation failed for any reason.

Syntax: **<var> = FILE_RENAME ('oldname.ext', 'newname.ext')**

Example: rc = file_rename ('e:\call.log', 'e:\oldcall.log')

Usage: The FILE_RENAME command allows you to change the name of a file. Again, the result of the rename function will be returned into the variable specified. The result will be a zero if the operation completed successfully, and a 1 if the operation failed for any reason.

Syntax: **<var> = FILE_CHGDIR ('folder')**

Example: rc = file_chgdir ('\flash')

Aliases: **FILE_CHDIR**

Usage: The FILE_CHGDIR command allows you to change the current default subdirectory (folder). This can be useful when preparing to run a GEM program that has a corresponding .RSC (resource) file that goes with it. For example:

```
script

integer rc

rc = file_chgdir ('\flash')
if rc > 0 then
   printe 'Sorry..unable to change directory!'
   exit
else
   gem 'flash.prg'
endif
```

Syntax: **TOS ('program.ext', 'command_line_parameters')**

Example: TOS ('c:\dos\arc.ttp', 'x program.arc *.*')

Usage: The TOS command operates like the GEM command, except it is used to execute a **.TOS** or a **.TTP** program. This is also the command that allows you to execute some of the external game programs written for BBS systems. **If the TOS command is used from a remote location, the program executed must handle all modem input/output and carrier checking itself!**

BBS Express! ST also passes certain information along to the TOS program on the command line. This information is passed in the following order:

user's handle
user's number
user's video width
time remaining this call
user's translation (0 = ASCII, 1 = Mono, 2 = Color)

Since the user's handle may contain spaces in it, the BBS will place character 160's in place of each space. It is up to your program to replace these 160's with spaces after pulling the name from the command line. This can be done in Personal Pascal in the following manner:

```
Cmd_GetArg (1, User_Handle)
For X := 1 to Length (User_Handle) do
   If User_Handle [X] = #160 then
        User_Handle [X] := ' ';
```

Also, note that these passed parameters are sent **after** any parameters that you specify on the TOS command. So, if you invoked an external program with the command:

TOS ('c:\games\blackjak.tos', 'one two three')

the actual command line passed to the program would look something like this:

c:\games\blackjak.tos one two three SYSOP*John 1 80 15 2

Syntax: **GEM ('program.ext', 'command_line_parameters')**

Example: gem ('ueditor.prg')

Usage: The GEM command allows you to execute a GEM program from within a script file. **This command should never be used from a remote location!** The reason it is included in the script processor is so that you may run GEM applications from the 'Waiting for call' screen.

IF Statement Processing

Syntax: **IF expression = | < | > | # expression [THEN]**

Example: If user_input = 'top menu' then

Usage: The IF statement is like that in any other language. It can check for equality (=), inequality (#), less than (<), or greater than (>). If the statement evaluates to TRUE, then all statements following this one up to:

- o an ELSE
- o an EF (elseif)
- o an ENDIF

will be executed.

Syntax: **IF COMMAND_LEVEL (number) [THEN]**

Example:
```
If command_level (32) then
   printe 'Greetings, SYSOP!'
Endif
```

Usage: This alternate version of the IF statement allows you to check the Command Security Level of the caller currently on-line, and take appropriate actions. The constraints of this statement are identical to the above IF statement.

Syntax: **IF MESSAGE_LEVEL (number) [THEN]**

Example:
```
If message_level (10) then
   printe 'You can read messages on base 10'
Endif
```

Usage: This alternate version of the IF statement returns true if the user can **read** messages on the message base number specified. This comes in very handy for menus displaying the available message bases on your board. For example:

```
script

printe 'Active Message Bases' center

if message_level (1) then
   printe '1.  General Messages'
else
   printe '1.  Private'
endif

if message_level (2) then
   printe '2.  Atari User Group'
else
   printe '2.  Private'
endif
```

Syntax: **IF FILE_LEVEL (number) [THEN]**

Example:
```
If file_level (5) then
   printe 'You can access file area number 5'
Endif
```

Usage: This alternate version of the IF statement returns true if the user can access the file SIG area specified.

Syntax: **IF TIME_EXCEEDED [THEN]**

Example:
```
If time_exceeded then
   printe 'Sorry..your time is up for today!'
   exit
Endif
```

Usage: Once a user is within a Script file, the BBS system no longer keeps checking whether the user has went over his/her time limit or not. The reason behind this is because you may need to perform some type of 'clean-up' in the script file, such as writing out a text file. You can do the time limit checking yourself using this command.

Syntax: **IF FILE_EXISTS ('filename.ext') [THEN]**

Example:
```
If file_exists ('e:\mainmenu.hlp') then
   view_file ('e:\mainmenu.hlp')
Endif
```

Usage: This command allows you to check on whether a specific file exists or not. If the specified file exists, then the IF statement will evaluate to true.

Syntax: **IF <var> = NUMERIC [THEN]**

Example:
```
If user_input = NUMERIC then
   goto carry_on
else
   printe 'Input a number, please!'
   exit
Endif
```

Usage: This IF statement will evaluate to true if the specified variable contains all numeric data (ie: it contains only the characters 0 - 9).

Syntax: **ELSE**

Example:
```
else
```

Usage: The ELSE statement will get executed if the previous IF or ELSEIF statement was false.

Syntax: **ELSEIF | EF**

Example: elseif

Usage: The ELSEIF command does what its name implies; it replaces a piece of code like:

```
if option = 'abort' then
   goto done
else
   if option = 'top' then
      goto menu
endif
```

with a simpler else check like this:

```
if option = 'abort' then
   goto done
ef option = 'top' then
   goto menu
endif
```

Syntax: **ENDIF**

Example: endif

Usage: An ENDIF statement terminates an IF statement. You should **always** have an ENDIF statement for **each** IF statement!

Syntax: **ABORT OFF | EXIT | label name**

Example: abort exit

Usage: The ABORT command allows you to control the actions taken if the user presses the ^C or ^X key while this script file is executing. This command can be used as many times as you wish within a single script file. **NOTE:** When a script file is initially invoked (or, when a new script file is EXECUTED), the default is set to ABORT EXIT.

If **OFF** is specified, then any ^C or ^X keys will simply be ignored.

If **EXIT** is specified, then the script file will be terminated immediately upon the user pressing the ^C or ^X key.

If a label name is specified, then the BBS will execute a **GOTO label** command if the user press ^C or ^X.

An example:

```
script
define instr
abort user_abort
top:
   clear (screen)
   dump
   this is a dump of lines
   this is a dump of lines
   this is a dump of lines
   end

        ... some more code ...

user_abort:
   printe
   printe '^C intercepted...quit? (y/n): '
   input instr 1 noreturn
   if instr = 'Y' then
      exit
   else
      goto top
   endif
```

Syntax: **EXECUTE ('filename.ext')**

Example: execute ('c:\bbs\scripts\script2.txt')

Usage: The EXECUTE command is used to execute another script file. The current script file is closed and the one named here is opened and processing continues. All work variables remain intact. You must give the entire filename to execute. The following happens when you EXECUTE another script file:

- All IF...ENDIF constraints are broken
- Any OPENed file is CLOSED.
- The ABORT flag is set to ABORT EXIT

Syntax: **EXIT**

Example: exit

Usage: The EXIT command immediately terminates the script file that is executing.

Syntax: **GOTO label**

Example: goto main

Aliases: **JUMP**

Usage: The GOTO statement will move you around in your script file. The processor will begin at the **top** of the script file looking for the specified label in the form of 'label:'. If found, execution will continue at the first line following the label. Also, note that you DO NOT put a colon (:) at the end of the label name in the goto statement (as you do with the label).

Syntax: **GOSUB label**

Example: **gosub any_key**

Aliases: **PERFORM**

Usage: The syntax of this command is equivalent to that of the GOTO statement. The difference in statements is that the GOSUB will return execution of the script file to the line following the GOSUB command upon encountering a RETURN command.

For example:

```
define  name
define  address
define  city
define  big_string

gosub get_3_parms
printe 'You Entered: [big_string]'
gosub get_3_parms
printe 'You Entered: [big_string]'
exit

get_3_parms:
        printe
        print 'Your Name: '
        input (name, 30, noreturn)
        print 'Your Address: '
        input (address, 30, noreturn)
        print 'Your City: '
        input (city, 30, noreturn)
        set big_str = '[name] [address] [city]'
        return
```

Syntax: **RETURN**

Example: return

Usage: The RETURN command returns control to the statement following the last executed GOSUB command.

Syntax: **FOR <var> = start_value TO end_value [DO]**

Example:
```
for count = 1 to 25 do
   printe 'Count = [count]'
endfor
```

Usage: With the FOR statement, it is possible to repeat a certain block of code a pre-determined number of times. The variable to the **left** of the equal sign is the 'index variable', and it will be increased by one each time the loop is executed. The code between the **FOR** and the **ENDFOR** statements will continue to be executed until the index variable is greater than the **end_value**.

Syntax: **ENDFOR**

Example:
```
for count = 1 to 25 do
   printe 'Count = [count]'
endfor
```

Usage: The ENDFOR statement marks the ending boundary of a FOR loop. You **must** have an **ENDFOR** statement for each **FOR** statement.

Syntax: **BREAK**

Example:
```
for count = 1 to 25 do
   if count > 20 then
      break
   endif
endfor
```

Usage: The BREAK statement can be used to prematurely exit out of a FOR..ENDFOR loop. When the break statement is encountered, execution will continue at the statement immediately following the ENDFOR statement. After BREAKing out of a loop, the index variable (COUNT, in the above example) will retain the value it had when the BREAK statement was encountered.

Miscellaneous Commands

Syntax: **CLEAR (SCREEN | TABLE)**

Example: clear (screen)

Usage: The CLEAR statement can be used to either clear the screen or to initialize the 100 table entries to a null value.

Syntax: **<dest_var> = COPY (<source_var>, start_pos, length)**

Example: substring = copy (string, 10, 5)

Usage: The COPY command is used to copy part of one string into another. The <dest_var> is set to the contents of the <source_var>, starting at position 'start_pos' in the string for a length of 'length' bytes. For example:

assuming STR1 equals 'BBS Express! ST', then

```
STR2 = Copy (Str1, 5, 8)
```

would result in STR2 containing 'Express!'

Syntax: **INCREASE <var> [amount]**

Example: increase counter

Usage: The INCREASE statement adds the specified AMOUNT to the variable named. If no AMOUNT is specified, then an AMOUNT of one is assumed. This is basically just a shorthand method of saying:

```
increase counter 15
```

instead of

```
counter = counter + 15
```

Syntax: **DECREASE <var> [amount]**

Example: decrease counter

Usage: The DECREASE command is identical to the INCREASE command, with the exception that the AMOUNT is subtracted from the specified variable.

Syntax: **DISCONNECT**

Example: disconnect

Usage: The DISCONNECT command will immediately hang up the current caller, resetting the BBS to the 'Waiting for Call' screen.

Syntax: **EXPAND (<var>, length)**

Example: expand (user_name, 60)

Usage: This command allows you to concatenate spaces onto the END of the specified variable. This is mainly used for formatting lines of output that have fields of varying sizes.

For example, the following code:

```
set f1 = 'ABCDE'
set f2 = 'FGH'
set f3 = 'ABCDEFGHI'
set f4 = 'JKLMNOP'
printe '[f1] [f2]'
printe '[f3] [f4]'
```

would yield this output:

```
ABCDE FGH
ABCDEFGHI JKLMNOP
```

whereas:

```
set f1 = 'ABCDE'
set f2 = 'FGH'
set f3 = 'ABCDEFGHI'
set f4 = 'JKLMNOP'
expand (f1, 15)
expand (f3, 15)
printe '[f1] [f2]'
printe '[f3] [f4]'
```

would yield:

```
ABCDE           FGH
ABCDEFGHI       JKLMNOP
```

Syntax: **<var1> = LENGTH (<var2>)**

Example: **lth = length (their_input)**

Usage: The LENGTH command will place into VAR1 the length of VAR2. For example:

assuming STR1 equals 'A string'

then

lth = length (STR1)

would set LTH equal to 8.

Syntax: **MARGIN (size)**

Example: margin (15)

Usage: The margin command is a useful tool for formatting PRINTEd lines dependent upon the user's terminal width setting. If a margin of greater than zero is active, that many spaces will be displayed before every PRINT/PRINTE command executed (unless the PRINT command has the CENTER option specified). Once a MARGIN setting is specified, it remains active until you specify another margin setting.

An example:

```
script
define  video_width =  '&11'     ;&11 is user's VW

if video_width = 40 then
   margin (8)
else
   margin (33)
endif

printe 'This should be centered.'
margin (0)
```

Syntax: **<var> = RANDOM (upper_limit)**

Example: counter = random (100)

Usage: The random command returns a number between one and 'upper_limit' into the specified variable.

Syntax: **[SET] <var> = 'string' | %nn | &nn [+-*/] [<var>]**

Example: **set prompt = 'Please Enter Your Name: '**
prompt = 'Please Enter Your Name: '

Usage: The SET statement is used to make on-the-fly changes to your work variables. You can do just about anything on the right side of the '=' sign. There can be either a string value, another work variable, or a user variable as the source of the SET. Also, **simple** numeric expressions may be specified on the right side of the '=' sign. In other words, if you wanted to multipy the variable COUNT times 2 and then add 50 to it:

```
count = count * 2 + 50          ;will NOT work

count = count * 2               ;will work
count = count + 50
```

It's even possible to concatenate strings. Let's say that you had the user's first and last name input separately, and you wanted to make them into the full name:

```
script
define first_name
define last_name
define full_name

print 'Your first name: '
input first_name 10 noreturn     ;get the first name

print 'Your last name: '
input last_name 10 noreturn      ;get the last name

set full_name = '[first_name] [last_name]'

printe 'Your full name is [full_name]'
```

Note that the word SET is an optional word that you can use or omit, depending on your personal preference. Also, note that when you concatenate strings, the source expression must be enclosed in quotes, and that variable names must be enclosed in brackets!

Syntax: **SHUFFLE <var1> <var2>**

Example: **shuffle cards suits**

Usage: Card games seem to be a popular use for script files. The SHUFFLE command makes these games much easier to write by giving you a fast way to 'shuffle' a deck of cards. Two variables are required by the shuffle command; variable one will contain the cards (in the form of A,2-9,T,J,Q,K) and variable two will contain the suits of these cards (in the form of S, H, D, C).

After the shuffle, the cards might look something like this (this is a 'perfect' shuffle):

```
cards = 'A23456789TJQKA23456789TJQKA23456789TJQKA23456789TJQK'
suits = 'sssssssssssshhhhhhhhhhhhhddddddddddddddccccccccccccc'
```

You can then select a card by use of the ARRAY_GET command. For example, to get the tenth card from the deck you could do the following:

```
the_card = array_get (cards, 10)
the_suit = array_get (suits, 10)
```

Syntax: **TIME_LIMIT (new_time_limit_in_minutes)**

Example: TIME_LIMIT (25)

Usage: The TIME_LIMIT command allows you to modify the current callers time limit (**for the current call only**). Using the TIME_LIMIT command **WILL NOT** change the 'time limit per call' in the user's record!

Here is exactly what happens: when a user first logs on, the BBS will set the 'time limit' to the value that is specified in their user record time_limit_per_call field. As long as you never execute a TIME_LIMIT command, then that will remain their time limit for this call. But, if you do modify their time limit with the TIME_LIMIT command, then that time limit will be used FOR THE CURRENT CALL ONLY, and it WILL NOT be written to their user record. The main reason behind this was a request to be able to shorten time limits during peak call periods each day. This is easily done with the TIME_LIMIT and &34 variable. Here is a sample script that you could place in your WELCOME.HLP file. It will set the caller to a time limit of 15 minutes if the current time is between 8:00 pm and 11:00 pm.

```
If &34 < '20:00:00'      ;don't do anything if
ef &34 > '22:59:59'      ; out of range
else
    time_limit (15)      ;it's in the range!
endif
```

Syntax: **<var1> = VALUE (<var2>)**

Example: **idx = value (their_input)**

Usage: The VALUE command can be used to change the first character of a string (<var2> above) into a numeric value. This numeric value is then stored in the variable to the left of the equal sign. For example:

assuming STR1 equals 'A'

then IDX = VALUE (STR1)

would set IDX equal to 65

Syntax: **<var> = VERSION**

Example: current_version = Version

Usage: The version command allows you to check the current version of the BBS that the user is running. This version will ALWAYS be returned as a four-digit number in the form of '0130' for version 1.30. This allows you to do numeric checks on the value returned. This is useful when writing script files that you plan to share with other people. If you, for example, have written your script file using commands that were not available in earlier versions of BBS Express! ST, you could do something like this:

```
the_version = version
if the_version < 130 then
    printe 'Sorry, this script file requires v1.30 or greater'
   exit
endif
```

Referencing Your Variables as Arrays

It is possible for you to access one (or all) of your work variables as an array of up to 100 elements of one character each. The ARRAY_xxx family of commands allow you to (1) pull elements out of the array, (2) store elements into the array, and (3) search for characters in the array. To illustrate this, let's take a look at the following example:

```
define mystr = 'Joe Smith'
```

Since all work variables are handled as strings, the variable MYSTR would look like this internally:

```
          J   o   e       S   m   i   t   h
Mystr   (1) (2) (3) (4) (5) (6) (7) (8) (9)
```

As you can see, a STRING variable is really nothing but an ARRAY [1..length] OF CHARACTER (in Pascal terms). The Array_xxxx commands allow you to access the individual characters of any string. Using the above example,

```
Mychar = Array_Get (Mystr, 5)
```

would set MYCHAR to the letter 'S' (which is the fifth letter in MYSTR).

Where is this useful? Well, it can be very useful anytime that you are writing a script file that needs to use a lot of 1-byte 'switches'. Let's say you had a script application that had a menu of 50 options for the user to select, but you wanted to ensure that they executed each option only one time. You could simply define a variable to hold these switches, and then use the ARRAY commands to mark and/or inquire on whether the user had executed an option. Here are some examples of how you would do this:

```
integer idx
array   switches        ;'0' = not yet executed, '1' = executed

;to initialize the array
for idx = 1 to 50 do
   array_put (switches, idx, '0')       ;set them all to not used
endfor

; to see if a user has selected this option
; (assume the number selected is in IDX)

temp = array_get (switches, idx)
if temp = '0' then
 ..they haven't yet done this..
else
   printe 'Sorry..you have already used that option!'
endif

; to mark an entry as have being used
array_put (switches, idx, '1')
```

One more example. In Pascal, you would use the following code to print out the elements of a string individually:

```
For X := 1 to length (Mystring) do
    Write (Mstring [X]);
```

The following SCRIPT code would accomplish the same thing:

```
Lth = Length (Mystring)
For X = 1 to Lth do
   Temp = Array_Get (Mystring, X)
   Write (Temp)
Endfor
```

The descriptions of the ARRAY commands follow:

Syntax: **ARRAY_CLEAR (<var>)**

Example: array_clear (mytable)

Usage: The array_clear command will set all 100 elements of the array specified to a null value. You should always clear an array before starting to use it, since 'left-over' values could pose problems in your code, especially if you are using the Array_Find command.

Syntax: **<target_var> = ARRAY_GET (<source_var>, index_into_array)**

Example: the_letter = array_get (mytable, 15)

Usage: The array_get command will set the target variable specified to the single letter contained in the source variable at the offset specified by the index_into_array field. For example:

assuming STR1 equals 'ABCDEFG'

then STR2 = Array_Get (STR1, 5)

would set STR2 equal to 'E'

Syntax: **ARRAY_PUT (<var>, index_into_array, 'character')**

Example: array_put (mytable, 10, 'A')

Usage: The array_put command will set the specified element of the named variable to the single letter specified as the third parameter (this can also be another variable).

Syntax: **<var1> = ARRAY_FIND (<var2>, 'character')**

Example: idx = array_find (mytable, 'A')

Usage: The array_find command will search the array specified by <var2> looking for the character requested. At completion, the index into the array at which the character was found will be placed into <var1>. If the character was not found in the array, then <var1> will equal zero.

Using the Dynamic Table

The **Table_xxx** commands operate much like the **Array_xxx** commands, except that the table commands operate on a 'built-in' table that contains entire strings (up to 100 bytes in length maximum). This table can be thought of as an additional 100 work variables. In Pascal terms, this table would be defined something like:

```
Var Table: Array [1..100] of String [100];
```

The main advantages of using the table are that (a) the **TABLE_LOAD** command makes loading data tables fast and easy, and (b) you can use the table entries as additional variables, with the indexing allowing a type of 'indirect addressing' without using a lot of code. The following example shows the little amount of code needed to get 10 input lines from the user:

```
script
define  instr                   ;what they enter in to
integer index                   ;work index

for index = 1 to 10 do          ;use the first 10 table entries
  input instr 70                ;get a line from the user
  table_put (index, instr)      ;stick it in the table
endfor
```

At the completion of the above piece of code, the 10 lines entered by the user are stored in the dynamic table in entries 1 through 10.

The syntax for the Table commands follow:

Syntax: **<target_var> = TABLE_GET (index_into_table)**

Example: the_word = table_get (15)

Usage: The table_get command will set the target variable specified to the table element specified by the index_into_table field.

For example assuming the table has the following in it:

(1) Joe Smith
(2) Barney Rubble
(3) Fred Flintstone

then:

```
MyStr = Table_Get (2)
```

would set MyStr equal to 'Barney Rubble'.

Syntax: **TABLE_PUT (index_into_table, 'string')**

Example: table_put (10, 'User Name: [users_name]')

Usage: The array_put command stores the string (or variable) specified into the dynamic table at the position specified by the index_into_table field.

Syntax: **<var> = TABLE_LOAD ('filename.ext')**

Example: total_entries = table_load ('e:\hangman.dat')

Usage: The table_load command will load the dynamic table from the text file specified. If the text file contains over 100 lines of data, then any lines over 100 will simply be ignored. The table_load command is an extremely fast way to load in data files, since it operates at the lowest level of GemDOS by doing block I/O. After the completion of the table_load, the number of entries (or lines) loaded into the table will be placed into the variable specified on the left side of the equal sign. This value effectively tells you where the end of valid table data is at. As an example, the following script file could be used to display text files (up to 100 lines in length):

```
script

define filename                          ;filename to here
define total_lines                       ;total lines read
define index                             ;work field
define temp                              ; same here

printe
print 'Enter name of file to view: '     ;get filename to view
input filename 40 noreturn

total_lines = table_load (filename)      ;load the table
for index = 1 to total_lines do          ;while still more
    temp = table_get (index)             ; get next line
    printe temp                          ;  and show it
endfor
```

Syntax: **<var> = TABLE_FIND ('string')**

Example: index = table_find (users_string)

Usage: The table_find command will search the dynamic table looking for a match on the string specified. If found, the <var> specified will be set to the index into the table at which the string was found. If the string was not found, then the <var> will equal zero. The comparison is **not** case-sensitive.

VT52 Color Commands

Syntax: **BACKGROUND (WHITE | RED | GREEN | BLACK)**

Example: background (green)

Usage: Sets the background color (for VT52 color users only) to the color specified.

Syntax: **FOREGROUND (WHITE | RED | GREEN | BLACK)**

Example: foreground (red)

Usage: Sets the foreground (character) color (for VT52 color users only) to the color specified.

Syntax: **INVERSE (ON | OFF)**

Example: inverse (on)

Usage: Turns reverse video on or off. (VT52 monochrome or color only.)

Word Parser Commands

The following parsing commands are intended mainly for the development of text adventure games with the Script language. These commands allow you to manipulate individual words in a whole string, making it possible to check for phrases like 'go north', 'take knife', etc. The syntax of these commands:

Syntax: **<var> = TOTAL_WORDS ('string')**

Example: count = total_words (user_input)

Usage: The TOTAL_WORDS command will return into <var> the number of words that are present in the string specified. Each word is terminated by a space. For example:

assuming STR1 equals 'This is my input line'

then count = total_words (str1)

would set count equal to 5.

Syntax: **<var> = GET_WORD ('string', word_number)**

Example: second_word = get_word (user_input, 2)

Usage: The GET_WORD command will set the <var> specified to an individual word from the string. The word returned is specified by the 'word_number' parameter. For example:

assuming STR1 equals 'This is my input line'

then word = get_word (STR1, 4)

would set WORD equal to 'input'.

Syntax: **<var> = POS ('substring', 'string')**

Example: index = pos ('goto', user_input)

Usage: The POS command will search the string specified to see if the substring specified is part of that string. If the substring is found to be present in the string, then the offset into the string will be returned into the variable specified. If the substring is not found, then <var> will return with a zero. For example:

assuming STR1 equals 'Take the knife and leave'

then index = pos ('knife', STR1)

would set index equal to 10.

Some Script File Examples

As you can see from above, writing script files really is just like writing a program. That is why we highly recommend that you study the example script files that are on your data disk. We have included an entire 'On-Line Order Entry System' written completely in script language. It is a good example of just how powerful this miniature language can be.

We are also going to provide you with some examples here that you can study. These script files are much simpler than some of the examples on your distribution disk, but they will help you to get an idea of exactly what you can and can't accomplish with script files.

By now, you have probably figured out that you don't need even a single **MENU** file; they could all be accomplished with **SCRIPT** files instead. Well, you are right! Which you should use is completely up to you.

As the first example, let's take one of the menu files that we studied earlier and convert it to a script file.

Here was what the menu file looked like:

```
menu return
Please enter your selection (or 'quit' to abort):
4
may¦c:\news\may.txt
june¦c:\news\june.txt
july¦c:\news\july.txt
quit¦exit

&00
```

Atari Club News

Which newsletter would you like to view?

May, June, or July?

The following script file would accomplish the same thing:

```
script

define  user_selection     ;define a variable

top:
          clear screen     ;clear the screen
          dump             ;show the menu to them

                                   Atari Club News

                          Which newsletter would you like to view?


                                   May, June, or July?

          end

retry:
          print 'Please enter your selection (or "QUIT" to abort): '
          input user_selection 4 noreturn          ;get their selection

          if user_selection = 'quit' then          ;they want out
              exit
          ef user_selection = 'may' then           ;if they entered a
              view 'c:\news\may.txt'               ; valid selection,
          ef user_selection = 'june' then          ; then view the
              view 'c:\news\june.txt'              ; appropriate file
          ef user_selection = 'july' then
              view 'c:\news\july.txt'
          else
              goto retry                           ;ask them again
          endif

          printe
          print 'Press <Return> to continue...'    ;pause here
          input user_selection 1   ;get a dummy input

          goto top                 ;and do it all again!
```

Also, there are many ways to write a script file to accomplish the same goal. Here is another way that the previous script file could have been written:

```
script

define  user_selection
define  path_name = 'c:\news\'     ;where the files are
define  filename                   ;file to be viewed

top:
          clear screen             ;clear the screen
          printe
          printe '              Atari Club News'
          printe
          printe ' Which newsletter would you like to view?'
          printe
          printe
          printe '            May, June, or July?'
          printe

retry:
          print 'Please enter your selection (or "QUIT" to abort): '
          input user_selection 4 noreturn          ;get their selection
          if user_selection = 'quit' then          ;they want out
              exit
          ef user_selection = 'may' then           ;if they entered a
              filename = 'may.txt'                 ; valid selection,
          ef user_selection = 'june' then          ; then set %5 to the
              filename = 'june.txt'                ; right filename.
          ef user_input = 'july' then
              filename = 'july.txt'
          else
              goto retry                           ;ask them again
          endif
          view '[path_name][filename]'             ;and show it to them

          printe
          print 'Press <Return> to continue...'    ;pause here
          input user_selection 1                   ;get a dummy input

          goto top                                 ;and do it all again!
```

In our last example, we are going to show you a script file that does file accesses. Some BBS systems allow users to leave a one-line message that will be viewed (at logon time) by the next user that calls the system. What we will do to accomplish this is to change the **LOGOFF.HLP** file to an intelligent script file that the user will invoke when he/she logs off of our system.

What we must do is outlined below:

- Ask the user if he/she would like to leave a message for the next caller.
- If they answer NO, then simply exit the script.
- If they answer YES, then we must:
    - Get an input line (the message) from the user
    - Write it to a file that gets displayed at logon time.

Here is a sample script file that would accomplish this:

```
script

define  In_Str              ;generic input string
define  Message             ;users 1-line message

printe
printe
printe 'Thanks for calling our BBS system!'
printe
printe 'Would you like to leave a one-line message for the next caller'
print  'to see?  (Y/N): '

input In_Str 1 noreturn

if In_Str = 'Y' then
   printe
   printe 'Enter your one-line message to the next user'
   print  '--> '
   input Message 78 noreturn                ;get their message

   open ('e:\data\message.txt', output)     ;open file as output
   fprinte                                  ;print blank line
   fprinte 'Message from &1:'               ;tell who it's from
   fprinte                                  ;another blank line
   fprinte Message                          ;and their message
   close                                    ;close the file
endif

printe
printe 'Please call back again real soon!'
```

To complete this example, we would have to show this file at some time during the logon process. The perfect place to do this would be in the **LOGSTAT.HLP** file, which is the last help file displayed when a caller logs onto the BBS. We simply change this file to a script file that does the following:

```
script

printe ' Online: &2.'
printe '  Alias: &1.'
printe '   From: &5, &6.'

printe ' Your Last Call Was On &12.'

printe ' Time Limit This Call: &13 Mins.'

view ('e:\data\message.txt')
```

The ENDCALL Script File

Every time a user logs off of the BBS, a special script file named **ENDCALL** is executed from your 80-column helpfile path. Using this script file, it is possible to write script files to do time-specific functions. A very useful example of this would be writing a script file to backup your RAMdisk files to the hard drive after the first caller of a new day hangs up.

This is really a simple task; the flow chart of the logic would go something like this:

- a one-line file is contained on the RAMdisk that contains the date of the last day we copied the files to the hard disk.

- When ENDCALL executes, all we need to do is to see if the date in that file is equal to the current date. If it is, then we simply exit the script file. If it is not equal, then we know that it is a new day. Then, we simply back up the files and write the new date to the file.

The script file on the following page would accomplish this (you'll have to change it for your own datasets, etc).

NOTE: Even if you don't plan on using the ENDCALL script, you should still place an empty ENDCALL.HLP file in your 80-column help path. If the ENDCALL file is not present, you will get System Errors logged to your call log dataset.

```
script

string  last_date
string  todays_date = '&24'        ;&24 = current date
string  log_message                ;logged to the call log
integer rc

  if FILE_EXISTS ('e:\datefile.dat') then ;only if the file is there!
     open ('e:\datefile.dat', input)
     finput last_date
  endif

EOF:               ;in case of empty file
  close
  if last_date = todays_date then          ;no need to carry on...
     exit
  endif

  printe 'Backing up datasets...'
  rc = file_copy('e:\filea.dat','c:\filea.dat')   ;back up this file
  if rc > 0 then
     log_message = 'Unable to backup filea.dat!'
     goto log_it           ;oops!
  endif

  rc = file_copy('e:\fileb.dat',c:\fileb.dat')    ;do next file
  if rc > 0 then
     log_message = 'Unable to backup fileb.dat!'
     goto log_it           ;oops!
  endif

  log_message = 'Files backed up successfully...'

  open ('e:\datefile.dat', Output)
  fprinte '[todays_date]'
  close
log_it:

  open ('e:\call.log', append)    ;write message to log file
  fprinte ('[todays_date] [log_message]')
  close
```

The Off-line Script Utilities

There are two programs on your BBS **Data disk** that make development of script files an easier task. These are the programs named **SCRIPT.TTP** and **SC.TTP**. The SCRIPT.TTP program is a 'Script processor emulator' that you can use to test your script files without having to boot the BBS. Both of these programs are designed to work with a Command Line Interface (commonly called a DOS Shell), but they can both be invoked from the desktop.

If SCRIPT.TTP is not passed a filename on the command line, then it will prompt you for both the filename to execute and the video width (40 or 80 columns) to emulate. SCRIPT.TTP can also be passed its parameters on the command line with the following syntax:

```
SCRIPT filename.scr video_width
```

where video_width is '40' or '80'.

SC.TTP is the off-line script compiler discussed earlier. Unlike SCRIPT.TTP, SC.TTP **requires** that the filename(s) to compile be passed on the command line, in the format of:

```
SC file1[.scr] [file2[.scr] file3[.scr] filen[.scr]]
```

As you can see, you can compile multiple script files at one time. If you have a script application that is made up of multiple script files (that get EXECUTEd) and **they share the same defined variables**, then all of the script files must be compiled together to work properly. This allows the variables to be shared across multiple script files.

Unless it is absolutely necessary (and, there really is no time that it should be absolutely necessary), it is recommended that you keep your script program in **one** file. This has many advantages, such as (1) it means less files taking up room on your disks, (2) it is easier to debug an application when all of the code resides together in one file, and (3) it is much easier for other sysops to use those script files that you wish to share (especially if they are not the script-writing type).

Running the BBS Program

Loading the BBS Program

The BBS program is on your distribution '**program**' disk under the name of **BBS.PRG**. If you are running from a hard drive, you should copy this file to the drive that you wish to boot from. Also, don't forget that the **BBS.PRG** file must be in the same folder that your **SYSDATA.DAT** file is in.

To load the BBS, simply double click on **BBS.PRG** from the GEM desktop, or key in BBS <return> if you are using a Command Line Interface.

After the BBS loads in (if you are booting from drive A: or B:) you will be prompted with

Insert Your Data Disks, Then Press Any Key...

If you are running from a floppy drive, then replace the disk in your floppy drive(s) with the appropriate data disks (the BBS.PRG file does not have to be present on any drive once it is loaded in).

The BBS will now load in all of the message bases so it can calculate the high message numbers for each individual base.

Next, the BBS will look for a file named **BOOTUP.MDM** on the default drive. If the file exists, the BBS will load in this text file one line at a time and send those lines directly to the modem. This file can be used to send any custom 'modem configuration' commands to the modem at boot-up time.

Next, the BBS will proceed to the 'Waiting for Call' screen. **Every** time this screen is invoked, the BBS will look for a file on the default drive named **RESET.MDM**. The BBS will search for this file every time a caller hangs up from the BBS. Again, if this file is found, the BBS will send the contents of it directly to the modem. If the file is not found, then the BBS will do its standard initialization, which consists of:

```
ATZ
ATS2=128 M0
```

Both the BOOTUP.MDM and RESET.MDM files will normally only be needed by those people running higher speed (2400-9600 baud) modems. The rule of thumb here is to first try the BBS without either one of these files present. Probably 95% of the modems available will work just fine with the BBS's standard configuration.

Adding Your Own Options To The 'Waiting for Call' Screen

After loading the BOOTUP.MDM file, the BBS will attempt to load a text file named **FUNCTION.DAT** (again, from the current drive and subdirectory). This file, if present, contains the literals that you want to appear on the 'Waiting for Call' screen for the F6 through F9 options. The F6, F7, F8, and F9 keys can be customized by you to load external programs, execute special script files, etc.

The process is very simple; when you press the F6 through F9 key, the BBS will attempt to execute a script file named Fx.HLP/COM/SCR from your **80-column helpfile path.** In other words, if you pressed the F6 key, the BBS would attempt to execute a file named F6.HLP/COM/SCR.

As an example, let's assume that we want to use the F6 key to execute a terminal program name SUPRTERM.PRG. All you need to do to accomplish this is as follows:

1. create a FUNCTION.DAT file for the description, such as:

   Run SuperTerm Program

2. create a script file in your 80-column pathname named F6.SCR. This script file would look like this:

   ```
   script
   GEM suprterm.prg
   ```

That's all there is to it! Now, the literal:

F6 - Run SuperTerm Program

will appear on your 'Waiting For Call' screen, and a simple press of the F6 key will load and execute the file SUPRTERM.PRG!

Options While Waiting For a Call

The waiting-for-call screen will always be invoked when the BBS is in an inactive state. You will note that there is a 'System Status:' area in the middle of the screen. It is on this line that any pertinent activity will be reported to you. At different times, any (or all) of the following will be indicated on this status line:

New Users - This means that there has been at least one user who has logged on as NEW since you last logged on to your system.

New Feedback - This means that someone has left 'Feedback to the SysOp' since you last logged on to your system.

New Uploads - This means that someone has uploaded a new file since you last logged on to your system.

At the bottom of the screen, you will see the options available to you at this point. All of these commands are invoked through use of the 'program function keys', or the F1 - F10 keys at the top of your keyboard.

F1 - Quick Entry, User #1	**F6** - N/A
F2 - Quick Entry, User #2	**F7** - N/A
F3 - Logon By User Name	**F8** - N/A
F4 - Normal User Logon	**F9** - N/A
F5 - Toggle Chat Mode	**F10** - Exit to TOS (Quit BBS)

The **F1 key** is the one that you should use almost all of the time to log on locally to your BBS. This logon method does the following: (1) it reads in user record #1, and then (2) proceeds directly to the 'Checking for Mail...' prompt.

The **F2 key** functions identically to the F1 key, except it reads in user record #2 instead of user record #1. The idea behind this is that you can use user #1 for your 'System Operator' ID, and use record #2 as your 'real name' ID. This is ideal for those situations where you want an ID that users don't know is in anyway associated with the System Operator.

The **F3 key** allows you to log on as another user by knowing only their user handle. Just press the F3 key, and you will be prompted to enter the user's name. Enter the handle of the user you wish to log on as, and the BBS will search for that user, read in his/her record, and then proceed to log you on as that user.

The **F4 key** will allow you to log on just as any remote caller would. When you run your BBS for the very first time, you will have to log on this way so that you can log on as NEW to get your user ID. Once you have gotten your ID (which must be #1), you can then use the USER EDITOR - EDIT USER CURRENTLY ONLINE option from the SysOp Command processor to change your record.

The **F5 key** will toggle CHAT MODE on and off, which will be reflected on the status line at the top of your screen.If you turn chat mode off, if a user

attempts to call for you, they will be informed that 'The SysOp is Not Around'. You will still be notified of this attempt, though, by the >>PAGED<< indicator on the status line.

The **F6 through F9 keys** are modifiable by you (see the **Adding Your Own Options To The 'Waiting for Call' Screen** section). If you have not specified a literal for a PF key in your FUNCTION.DAT file, then that key will show up as 'N/A' (not applicable).

Options While A Caller Is Online

While a caller is online, you (SysOp) can use your keyboard to key anything in, just as though the caller had pressed the keys. In addition, there are also the following special key presses:

ESC key - The escape key is used to enter and exit **Chat mode.**

Shift F10 key - Pressing the **Shift-F10** key will immediately hang the caller up. It simply fakes the BBS into thinking that the carrier has been dropped.

$** - If you press the '$' key from the MAIN COMMAND prompt (or the key defined to be the SysOp mode command in your SYSDATA file) while a caller is online, the following will occur. You will be asked if you want the caller that is online to see what you are doing. If you answer YES, then the caller will be able to watch you doing your work, but they will not be able to press any keys. If you answer NO, then the user will receive the message **'The SysOp is working...be patient' and the caller will be 'frozen' until you exit out of SysOp mode.

NOTE: If you press '$' from the sysop keyboard, the BBS will **ALWAYS** enter Sysop Command mode, regardless of the security level of the user online.

Command Overview

Following is a command summary of each default command available to the user under BBS Express! ST. Note that you may change any of the keypresses used to invoke the main and message base commands by modifying your SYSDATA.DAT file.

From the Main Command Prompt

B The 'B' command is used to Browse Download Files. The user will be prompted for the file area that they wish to enter. If the user enters a '0', they will be shown the valid file SIG areas by name (as defined in your SYSDATA file). If a user does not have access ability to a particular SIG, it will simply be displayed as 'Private'.

C The 'C' command is used to 'Call for the SysOp'. When a user presses this key, you will hear 20 'bells' on your terminal, which tells you that someone wishes to chat with you. You enter and exit chat mode with the ESCape key. If you do not answer the call immediately, the word '>>PAGED<<' will appear on the status line at the top of your screen. If you have chat turned OFF, then the user will be told that you are not available for chat if they attempt to page you (but, the status line will still show '>>PAGED<<').

F The 'F' command allows the user to 'Leave Feedback to the SysOp'. Depending on your SYSDATA setting, this letter will either go into the Electronic Mail base (#32), or to the printer.

G The 'G' command stands for 'Goodbye'; it simply logs the caller off (after displaying the file LOGOFF.HLP to them).

I The 'I' command is the 'Info on this System' command. The help file that this command invokes is SYSINFO.HLP. Use these files to tell the user something about the software and hardware that your BBS system is running on.

L The 'L' command is the 'Enter Library' command. The help file invoked is LIBRARY.HLP. You can use MENU files in this area to maintain a collection of text files.

N The 'N' command, 'Read System News', can be used by the caller to read the SYSNEWS.HLP file.

O The 'O' command, 'Other BBS Numbers', displays the help file named BBSLIST.HLP. This is a text file containing names and numbers of other known BBS systems.

P The 'P' command, 'Edit Your Parameters', is a very important one for your users to understand. With this command, the user can change many aspects of their activities with BBS Express!, such as whether they can display color, whether they want the screen cleared after each message displayed, etc. This command can also be used by the caller to change their name (NOT their handle), address, city, state, country, etc.

Q The 'Q' command, 'Quickscan All Marked Boards', will be the one used most on your BBS. This command allows the user to go to each message base that they wish to access (specified from the 'P' command above) and read all new messages on that base (all from one keypress!).

R The 'R' command, 'Read Electronic Mail', can be used by the caller to display any Email that has been left for them. This command is automatically invoked by the BBS when the caller first logs onto the system each time.

S The 'S' command, 'Send Electronic Mail', is for sending Email to another user. The caller will be prompted for the handle of the person that they wish to send mail to. After getting the name from the caller, the BBS will check the userlog to make sure that such a user does exist.

T The 'T' command, 'View Title Page', can be used by the caller to re-view the WELCOME.HLP file.

U The 'U' command, 'Users of this BBS', will show the userlog of your BBS to the current caller. They will be shown the user's handle, city, state, and last call date. If you don't want this command included in your BBS, simply use the SYSEDIT program and change the 'U' to invoke this command to a '-'.

W The 'W' command, 'Who's Called?', will show information on the last 50 callers to your BBS. This information is kept in the SYSDATA.DAT file, and never has to be maintained by you.

Y The 'Y' command, 'Your Statistics', views the USERSTAT.HLP file. Use this file to display pertinent information to the user, such as their name, handle, user number, etc.

***** The '*' command, 'Message Base List', displays the file BASELIST.HLP. Use this file to tell the user what message base areas are available on your BBS.

= The '=' command, 'Go To A Message Base', is how the user moves around between message bases. The user will be prompted for the number of the message base to enter. **NOTE:** A level 32 user can do an '=32' command and view the Email base just like it was a normal public message base.

/ The '/' command, 'Go To the First Base w/new Messages', is an alternative manual-mode quickscan. The BBS will move the user to the first message base that has messages new to them.

**$** The '$' command allows you to enter the SysOp command mode (explained in detail below).

D On your sample BBS, there is a 'D' command included which is a version of the 'On-Line Order Entry' system that we use on our BBS at ICD. This is a good example of just how powerful Script files can be.

From the Message Base Prompt

B The 'B' command, 'Read Backward Sequentially', allows the user to read messages in reverse order.

C The 'C' command, 'Clear All Message Markers', allows the user to clear any messages that they have previously marked for reading. This is a good command to do before scanning and marking messages by titles.

D The 'D' command, 'Delete a Message You Wrote', can be used by the caller to delete a message they had previously posted. Only the person who wrote a message can delete it (unless, of course, they have the 'DELETE ANY?' security flag set in their user record).

E The 'E' command, 'Edit a Message You Wrote', allows the user to edit a message that they had previously saved. From that point on, the word '[* Edited *]' will appear in the message header of that message. Only the person who wrote the message can edit it (unless they have the 'EDIT ANY?' security flag set in their user record).

F The 'F' command, 'Read Forward Sequentially', allows the user to read messages in forward order.

K The 'K' command, 'Read Messages by Keywords', allows the user to read messages depending on certain data matches. The options are to search by Author, Title, Time, Date, the To User, and even Message Data!

N The 'N' command, 'Read New Messages', will display to the user those messages which are new to him/her. This command is invoked by the BBS automatically if the user is 'Q'uickscanning.

P The 'P' command, 'Post a Message', allows the user to write a new message on the current base. They will be prompted for a message title, along with who the message is intended for.

Q The 'Q' command, 'Add/Remove Board from Qscan', allows the user to add or remove the current message base from his/her quickscan list.

R The 'R' command, 'Read an Individual Message', allows the user to read a single message. The user will be prompted for the message number to display.

S The 'S' command, 'Scan Titles with Read/Mark', allows the user to see the title of each message on the current message base. In addition, the user can also mark that message for later viewing, or read it immediately.

T The 'T' command, 'Title Listing', displays the same information as the 'S' command does. The difference is that there is no pause for actions by the user.

X The 'X' command, 'Exit To Main Command', is used to return to the main command prompt.

Y The 'Y' command, 'Read Messages to You', will display to the user all messages that are addressed to him/her.

= The '=' command functions identically to the one from the main command prompt.

/ The '/' command functions identically to the one from the main command prompt.

From the File SIG Prompt

B The 'B' command, 'Browse Files', will display files to the user with file descriptions included.

C The 'C' command, 'Catalog Listing', is a short form of the display files command. File descriptions are not displayed.

D The 'D' command, 'Download a File', allows the user to begin a file transfer from the SIG command prompt. The user will be prompted for the filename.

U The 'U' command, 'Upload a File', allows the user to begin a file upload. The user will be prompted for the filename. After the file transfer is complete, the user will be forced into the message editor to enter a file description for this file.

S The 'S' command, 'Browse Files by Date Specifier', allows the user to display files that have a date greater than or equal to the one they enter.

N The 'N' command, 'Browse New Files', allows the user to see files that have been uploaded since their last call. This is in effect simply a 'S' command with today's date.

X The 'X' command, 'Exit to Main Command, allows the user to exit this file SIG and return to the main command prompt.

/ The '/' command, 'Go To Next File Area', will place the user into the next sequentially numbered file SIG area.

G The 'G' command, 'Goodbye', will log the user off.

From the SysOp Command Processor

DIR Gives you a disk directory. If no parameters are specified, then the files on the current default drive/folder will be displayed.

GEM Allows you to execute an external GEM-based .PRG program.

TOS Allows you to execute an external .TOS or .TTP program.

TIME Used to set the current time (based on a 24 hour clock). An example is TIME 14:42 to set the time to 2:42 pm. If no time is specified, then the current time will be displayed.

DATE Used to set the current date. An example is DATE 7/14/87 to set the date to July 14, 1987. If no date is specified, then the current date is displayed.

COPY Used to copy files. Wildcards are allowed in the copy. Some examples:

```
copy a:file1 b:file2
copy c:\menus_40\*.* d:\holdem\menus_40\*.*
copy c:\bases\ d:\bases\
```

If no filespec is entered (as in the last example above), the BBS will append '*.*' to both filenames.

CHAT Used to set chat mode ON or OFF. If no parameter is specified, then the current setting of chat mode will be displayed.

VIEW Used to view a text file.

EXECUTE Used to invoke a script file, menu file, or survey file. Useful if you want to test a file before editing your SYSDATA file to put it 'on-line' to users.

EDIT Allows you to use the message editor of BBS Express! to edit a text file. Note that you only have 50 lines to work with! If you load in a file bigger than 50 lines and then /Save it, you will lose part of the file!

EXIT Exit Sysop mode and return to the main command prompt.

FREE Show freespace on the specified drive.

MEMORY Show available memory (not very useful - this was more of a debugging tool for the author).

UNARC List the contents of the specified ARC file.

TOUCH Allows you to update the timestamp and datestamp of a file.

CREDIR Allows you to create a new subdirectory (folder).

CHDIR Allows you to change your current working subdirectory (folder).

DELDIR Allows you to delete a subdirectory (folder). The folder must be empty (contain no files) for the delete to work.

PRINT Print the specified text file to the printer.

RENAME Rename a file. Wildcards are NOT allowed on a rename. Also, don't forget that you can 'move' a file by renaming it to a different subdirectory, so be careful! As an example, the following command:

```
rename menus_80\mainmenu.hlp menus_40\mainmenu.hlp
```

would MOVE the file mainmenu.hlp from one folder to the other!

DELETE Used to delete a file. Wildcards are allowed. You will be prompted to verify the delete of each filename.

HELP Displays the file SYSOPCMD.HLP.

CALL LOG If your call log is going to disk, this command allows you to view its contents.

NEW FILES Used to browse all new files that have been uploaded. The BBS will scan all file SIGS and display new files to you, which you can then validate, etc.

USER EDITOR Invokes the on-line user editor. The most important thing to note here is that if you are editing the user that is currently on-line, then BE SURE to use the 'Edit User Currently On-Line' option. If you were to read in their record 'by user number or name', and edit it, then the record would be over-written with the same old information when the user logged off from this call.

The Off-line User Editor

While the on-line user editor in the BBS is sufficient for almost all of your needs, there are going to be times when you will need the off-line user editor. The off-line user editor has some very useful utilities such as

- Delete users by last call date
- Mass reset of survey bits
- Extend your USERLOG file allocations
- Edit users from a formatted list

The off-line editor is a GEM-based application, so spending much time here explaining it is unnecessary. Each function is clearly grouped under the different menu bar titles.

To use the off-line editor, double click on the file **UEDITOR.PRG** from the GEM desktop. Once the file is loaded, you will need to move the mouse to the 'File' entry on the menu bar, and then click on the '**Open...**' entry. You will be presented with the standard GEM file selector box, which you should use to tell the program where your userlog file is.

Once you have opened your userlog file, you have the following options available to you:

'Utilities'

Formatted List

The formatted list option will show you a list of each user in your userlog database. Use the arrow keys to scroll the list up or down. To edit a user, just move the arrow to either the user's name or handle and click once with the mouse button.

Delete by Date

This option allows you to delete the records of callers who haven't called your BBS since the date specified. This is handy for running periodic purges of unused records. Most SysOps will delete any user who hasn't called in the last 90 days.

Convert to Upper/Lower

This utility will go through each record in your userlog and convert each text field to a mixture of upper and lower case. For example, JOHN E. SMITH would be changed to John E. Smith.

Mass Sec. Lvl Change

By using this option, you can make mass changes to individual security bits in each users records. You must choose whether you want to (1) turn ON (set) bits, or (2) turn OFF (clear) bits. Also, you must specify the security flag that is to be modified. Next, you must place a checkmark

under each bit that you want to be turned ON or OFF (specified by 1 and 2 above).

Reset Surveys

When you change a survey, you must change all of the user records to indicate that each user has not yet taken this survey. This is done with the Reset Survey Bits command. You will be shown a screen with 32 boxes; just click on the corresponding box for the survey number that you want to be reset.

Extend Userlog

This utility is what you use if you should ever need to make your userlog file bigger (to allow more user records). You will be shown the number of records allocated and be prompted to key in the number of records you would like it to be changed to. You can only extend upward (in other words, you can't make your userlog smaller). **PLEASE BACK UP YOUR USERLOG BEFORE USING THIS FUNCTION!**

'Add User'

Use First Available Slot

This is the preferred way to add a new user. The userlog will be searched looking for a 'deleted' user record. If one is found, you will edit that record. If a deleted record is not found, then you will edit the next available record at the end of the database.

Add to End of Database

This function allows you to add a new user record directly to the end of the current database.

'Print'

Long Listing

This command will print a 7-line-per-user report to your printer. It shows all pertinent information on each user.

Short Listing

This command will print a 1-line-per-user report to your printer. It shows each user's handle, password, city, and state.

Align Labels

This option will print a template of two mailing labels to your printer. It is useful for lining up your mailing label sheets before printing the actual mailing labels.

Mailing Labels

This option allows you to print mailing labels for each user in your Userlog database. The program expects you to use mailing labels that measure 3 1/2" x 15/16", one across.

'Edit User'

by Data Search

After choosing this function, you will be shown a box in which you can enter many different text fields (such as name, handle, city, state, etc). Fill in any data that you wish to be included in the search criteria. After clicking on the OK box, the userlog will be searched looking for partial (or full) matches on any of the data that you just entered in. On any match that is found, you will be shown that user's name and handle and be given the chance to edit the record.

by User Number

This function is the quickest way to edit a user (if you know what that user's number is). You will be prompted for the record number to edit.

by New User Flag

You can use this function to edit all of those users that have logged on as **NEW**, but have not yet been edited by you. Once you save a new user's record, that record is no longer considered new.

Visitor Record

Use this function to edit the **Visitor Record.** When a user logs onto your BBS as a 'visitor', their user record will be set to an exact image of this record. So, the visitor's security levels, time limits, etc. are kept in this record. **YOU SHOULD EDIT THIS RECORD BEFORE PLACING YOUR BBS ON-LINE FOR THE FIRST TIME.**

New User Record

Use this function to edit the **New User Record.** When a user logs onto your BBS as a 'NEW USER', their user record will be set to an exact image of this record. So, the new user's security levels, time limits, etc. are kept in this record. **YOU SHOULD EDIT THIS RECORD BEFORE PLACING YOUR BBS ON-LINE FOR THE FIRST TIME.**

Understanding Security Levels for BBS Express! ST

With BBS Express! ST, you have COMPLETE control over almost every action that a user can or can't perform. As you have probably noticed by now, everything under BBS Express! ST is broken up into 32's; there are possible maximums of 32 message bases, 32 file SIG areas, 32 trackable surveys, and 32 command levels.

So, it should come as no surprise that each user's record has groups of 32 on/off flags for different functions. These possible functions are broken up in the following manner.

For the message bases, there are 32 individual flags for:

- Can this user READ messages on this base?
- Can this user POST messages on this base?
- Can this user EDIT *ANY* message on this base? (co-sysop function)
- Can this user DELETE *ANY* message on this base? (co-sysop function)
- Can this user PRINT *ANY* message on this base? (co-sysop function)

For the file areas, there are 32 individual flags for:

- Can this user ACCESS this SIG?
- Can this user VALIDATE files on this SIG? (co-sysop function)
- Can this user EDIT DESCRIPTIONS on this SIG? (co-sysop function)
- Can this user DELETE *ANY* file on this SIG? (co-sysop function)

For the command security, there are also 32 individual flags to signify whether a user may execute that level of a command.

As an example, let's assume that we have a user named Joe Smith that has the following security level flags:

```
COMMAND:                          1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3
              1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2

Commands      Y Y . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

MESSAGE BASE:                     1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3
              1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2

Read Msgs?    Y Y Y Y . . Y . Y . . . . . Y . . . . . . . . . . . . . . . . . Y
Post Msgs?    Y Y . . . . Y . . . . . . . Y . . . . . . . . . . . . . . . . . Y
Edit Msgs?    . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
Del  Msgs?    . . . . . . . . . . . . . . . Y . . . . . . . . . . . . . . . . .
Print Msgs?   . . . . . . . . . . . . . . . Y . . . . . . . . . . . . . . . . .

FILE SIGS:                        1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 3 3 3
              1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2

Access SIG?   Y Y Y Y . . . . . . . . . . Y . . . . . . . . . . . . . . . . . .
Validate?     . . . . . . . . . . . . . . Y . . . . . . . . . . . . . . . . . .
Edit Desc?    . . . . . . . . . . . . . . Y . . . . . . . . . . . . . . . . . .
Del Files?    . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

Now, just what exactly can Joe do? Well, as far as command levels, he can execute level 1 and 2 commands only. Remember, command levels for each individual command are kept in your SYSDATA.DAT file, and can be changed by using the SYSEDIT program. This is probably the right time to explain the importance of not giving people 'SysOp Access'. First, you should ALWAYS keep the security level of the 'SysOp Command Mode' (in your SYSDATA file) at level 32.

Next, you should NEVER give any of your users level 32 command access unless you are perfectly sure of what you are doing. Why? Well, the reasons are simple. The 'Sysop Command Processor' area of BBS Express! is just about as powerful as any 'DOS Shell' that you can buy. From within the SysOp processor, it is possible to delete every user in your user log, delete any files on any of your drives, delete any subdirectory, etc. So, as you can see, a terribly upset level 32 Sysop could wipe out your ENTIRE hard drive in a matter of minutes (after all, a 'DELETE *.*' command doesn't take long to execute!).

As for the message bases on our BBS, Joe can do the following. He can READ messages on bases 1, 2, 3, 4, 7, 9, 15, and Electronic Mail (#32) (if a user can not read messages on a base, they will be greeted with an 'invalid message base' message if they try to enter that base). Joe can POST (write) messages on message bases 1, 2, 7, 15, and Electronic Mail. Thus, it's possible (and sometimes very convenient) to allow a user to read messages on a message base, without allowing them to write messages or reply to existing messages. This is sometimes the perfect set-up for 'NEW' users.

Not long ago, Joe came to us with a request. You see, Joe is a MIDI music addict, and he asked us if we would add a new message base and file SIG area devoted totally to MIDI music. Our reply? Well, as any good SysOp would do, we said 'Sure, Joe...as long as YOU volunteer to maintain it! You will have to police the message bases for any vulgar messages. You will also have to periodically validate new upload files, and edit any descriptions that you think could be improved'.

Well, Joe agreed to be the CO-Sysop of the MIDI Music areas. So, we set up a new message base and new file SIG, both as #15. We then updated Joe's security levels (which were shown above). If you will look at Joe's message base flags under base 15, you will see that he can do just about everything that is possible on a message base. He can read messages, post messages, delete messages, and print messages (in case he wants you to see one). But, he can't edit messages. While we like Joe, we don't want him editing other people's messages and possibly changing something that is said (this is probably unrealistic, but used mostly for demonstration purposes).

Onto the File SIGS. Joe can only access file areas 1, 2, 3, 4, and 15. Once again, he has much more power in the #15 file area. There, he can also validate new upload files and edit any file description. He does not have the ability to delete any file on that SIG; we would prefer do that on our own.

Setting Up Your New User and Visitor Record

Now that you understand how security levels control each user's actions, we can discuss two VERY important 'user records' in your userlog file. These records are referred to as the NEW USER RECORD and the VISITOR RECORD. These records are just like any other userlog data record, except that they are solely for your use, and no user can ever log on with them.

Their importance is that they control what happens when a NEW user logs onto your board (either as a permanent user or a visitor). Here is what BBS Express! ST does: when a user logs onto your board as NEW, Express! will ask them if they want a permanent password. If they respond with 'Yes', then the NEW USER RECORD contents will be copied into their user record. If they respond with 'No', then the VISITOR RECORD contents will be copied into their user record area. So, these records become the base that their records are built upon.

Not every field in the NEW USER and VISITOR records are important. For example, any data that the user is prompted for (such as handle, name, address, etc) will overwrite what was in the new user/visitor record. The most important fields in the new/visitor records are the security levels and the time limits. Here is a list of the fields that you SHOULD update in your new user and visitor records:

Time limit per call
Time limit per day
Download ratio

Command Security Level
Message Base security level
File SIG security level

It is recommended that you NEVER allow VISITORS to post any messages; just let them read messages and look around your BBS.

Fine!

As you can see by now, BBS Express! ST is more than just another BBS. Thanks to our SCRIPT file language, it can be used as a remote order entry system, to provide online games, and allow custom menus, giving the SysOp total freedom to be creative.

Check with our support BBS from time to time for utilities and SCRIPT files from ICD and other BBS Express! ST supporters.

If there are any additions you would like to see to this manual, please drop us a note. Thank you again for supporting our products.